For Reference

NOT TO BE TAKEN FROM THIS ROOM

Ex libris
universitatis
albertaensis

HIGH LEVEL BOOK
BINDERY LTD.
10372 - 60 Ave., Edmonton
"THE HIGHEST LEVEL OF
CRAFTSMANSHIP"

Digitized by the Internet Archive
in 2020 with funding from
University of Alberta Libraries

https://archive.org/details/Dimsdale1971

THE UNIVERSITY OF ALBERTA

APPLICATION OF ON-LINE COMPUTER SYSTEMS TO LIBRARY AUTOMATION

by

© JAMES JOHN DIMSDALE

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1971

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Application of On-Line Computer Systems to Library Automation" submitted by James John Dimsdale in partial fulfilment of the requirements for the degree of Master of Science.

Date August 16, 1971

ABSTRACT

The investigation reported in this thesis was composed of two parts. The first part involved the design, implementation, testing, and evaluation of an integrated library automation system that encompasses an on-line catalog subsystem and a real-time circulation subsystem. A detailed discussion of this library automation system, called the IT System, forms the main content of this thesis. The second part of the investigation involved the design of a computer file structure that could support an on-line integrated library automation system that would serve the needs of a large academic library. The proposed design for this file structure is confined to the files that are necessary for the support of an on-line catalog subsystem and a real-time circulation subsystem. It represents an attempt to design a file structure that is not only economical in terms of storage requirements and in terms of CPU time needed to maintain and search it, but is also capable of providing very short response times for most on-line transactions and queries. Most of the effort in this phase of the investigation was expended in the design of a new method for the construction of the inverted index files of the on-line catalog subsystem. The exposition of this method, which is based on the principles of virtual hash addressing, covers the underlying theory, the detailed structures of the index files for authors, titles, and Library of Congress call

numbers, the procedures that are used in the generation, maintenance, and search of these files, and the theoretical performance of these files in terms of storage requirements and file access times.

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Professor H.S. Heaps, my supervisor, for his advice, guidance, and financial support during the preparation of this thesis; to Professor D.M. Heaps for her continuing interest and encouragement; to Miss Janice Heyworth for her many helpful suggestions and for her assistance in the testing and evaluation of the IT System; and to Mrs. Isabel Bince who typed and reproduced this thesis under trying circumstances. I am also indebted to Bill Lee who programmed the initial version of the IT System. Above all I would like to thank my wife, Christine, to whom this thesis is dedicated, for her patience, understanding, and constant encouragement.

TABLE OF CONTENTS

Page

CHAPTER I: INTRODUCTION

1.1 The need for Automated Library Systems. . 1
1.2 Evolution of Single Task and Integrated Systems 3
1.3 Off-Line Single Task Systems 6
1.4 On-Line Single Task Systems 10
1.5 Off-Line Total Systems 13
1.6 On-Line Total Systems 14
1.7 Statement of the Problem 19

CHAPTER II: REVIEW OF THE SALIENT LITERATURE RELATING TO ON-LINE LIBRARY AUTOMATION SYSTEMS

2.1 Introduction 24
2.2 The LOLA Subsystem 24
2.2.1 Overview of LOLA 24
2.2.2 LOLA's Disk Files 25
2.2.3 Data Capture and Flow in the Integrated System 26
2.2.4 Search of the In-Process File . 26
2.2.5 File Organization and Record Structures 28
2.2.6 Evaluation 28
2.3 The BELLREL System 29
2.3.1 Introduction 29
2.3.2 Overview of BELLREL 29
2.3.3 File Organization and Record Structures 30
2.3.4 On-Line Transactions 32
2.3.5 Batch Processes and Products . . 34
2.3.6 Evaluation 34
2.4 The Berkeley Bibliographic File Organization Project 35
2.4.1 Introduction 35
2.4.2 Hardware 35
2.4.3 Software 35
2.4.4 File Organization 37
2.4.5 Bibliographic Record 39
2.4.6 Conversion of Existing Records . 39
2.4.7 Evaluation 39
2.5 Project INTREX 40
2.5.1 Introduction 40
2.5.2 The Data Base and the Augmented Catalog Concept 41
2.5.3 Equipment 42

Page

2.5.4 Subject Indexing 42
2.5.5 File Organization 43
2.5.6 Catalog Records 43
2.5.7 Inverted File Records 44
2.5.8 Catalog Directory Records 45
2.5.9 Searching the Augmented Catalog . 45
2.5.10 Evaluation 46
2.6 Summary 47

CHAPTER III: THE IT SYSTEM AND ITS COMMAND LANGUAGE

3.1 Introduction 48
3.2 Data Base 48
3.3 Equipment 49
3.4 Overview of the Command Language 50
3.5 Definition of Terminology 50
3.5.1 Definition of the Meta Language Used to Describe the Command Language 50
3.5.2 Basic Definitions for the Command Language 52
3.6 The User-Oriented Commands 55
3.7 The Circulation Commands 60
3.8 The Miscellaneous Commands 65
3.9 Prompting 66

CHAPTER IV: THE FILE ORGANIZATION AND RECORD STRUCTURES OF THE IT SYSTEM

4.1 Introduction 69
4.2 The Catalog and Loans Group 72
4.2.1 The Catalog File 72
4.2.2 The Loans File 80
4.2.3 The Author Index File 82
4.3 The Patron Group 84
4.3.1 The File USERS 86
4.3.2 The User Index Files and Tables . 92
4.3.3 The File OVERDUES 94
4.4 The Patron Auxiliary Group 95
4.4.1 The File FREESTOR 95
4.4.2 The File FREELOCS 96
4.5 The Catalog and Loans Maintenance Group . 97
4.5.1 The File NEWBOOKS 97
4.5.2 The File NEWENTRIES 104
4.5.3 The File SAVEFILE 104
4.5.4 The File DELETIONS 105
4.6 The Patron Maintenance Group 106
4.7 The File OFFLINE.I/O 114

CHAPTER V: THE PROGRAMS OF THE IT SYSTEM

5.1 Introduction 117

Page

5.2 The On-Line Module LIBRARY 117
5.3 The Off-Line Module BATCHUPDATE 120
5.4 The Off-Line Module PICKUP 123
5.5 The Off-Line Module BATHENROLL 123
5.6 The On-Line Module ONLINEUPDATE 123

CHAPTER VI: PROPOSED FILE STRUCTURE FOR THE ON-LINE CATALOG SUBSYSTEM

6.1 Introduction 125
6.1.1 Brief Review of Work Done on the Design of File Organizations for the On-Line Catalog Subsystem . 127
6.1.2 Criteria for the Design of the On-Line Catalog Subsystem . . . 128
6.1.3 Overview of the File Organization of the On-Line Catalog Subsystem. 130
6.2 The Theory of Virtual Hash Addressing . . 133
6.2.1 Introduction 133
6.2.2 Hash Addressing 133
6.2.3 Collisions at Real Addresses . . 136
6.2.4 Virtual Hash Addressing 138
6.2.5 Collisions at Virtual Addresses . 140
6.2.6 Determination of the Number of Bits Needed in the Virtual Hash Address 140
6.3 Basic Structure of the Hash Index Files . 143
6.4 Theoretical Considerations Related to the Choice of Bucket Capacity 148
6.5 Insertion of Keys into the Hash Index Files 151
6.5.1 Introduction 151
6.5.2 The Simplest Case of Key Insertion 151
6.5.3 The Method Used to Handle Collisions at Real Addresses . 153
6.5.4 The Method Used to Handle Collisions at Virtual Addresses 155
6.5.5 The Methods Used to Handle Bucket Overflows 155
6.5.6 Algorithms for the Selection of Overflow Buckets 160
6.5.7 Insertion of Keys When Bucket Overflows are Taken Into Consideration 165
6.5.8 The Displacement Algorithm . . . 166
6.6 The Retrieval of Keys and Associated Data From the Hash Index Files 167
6.6.1 The Search Algorithm 167
6.6.2 The Mismatch Problem 172
6.7 Frequency Loading of Keys 173
6.8 The Title Word Hash Index File 174

Page

6.8.1 Estimate of the Number of Different Title Words in One Million Titles 174
6.8.2 Determination of the Number of Bits Needed in the Major, Minor, and Complete Virtual Hash Address . 175
6.8.3 Structure of the Title Word Hash Index File 176
6.8.4 Number of Content Entries Needed In Each Bucket 177
6.8.5 Bucket Size 177
6.8.6 Size of the Title Word Hash Index File 178
6.8.7 Expected Time to Access a Bucket. 178
6.9 The LC Call Number Hash Index File . . . 179
6.9.1 Number of Bits Needed in the Major, Minor, and Complete Virtual Hash Address 179
6.9.2 Structure of the LC Call Number Hash Index 179
6.9.3 Number of Content Entries Needed In Each Bucket 180
6.9.4 Bucket Size 180
6.9.5 Size of the LC Call Number Hash Index File 181
6.9.6 Expected Time to Access a Bucket. 181
6.10 The Author Hash Index File 181
6.10.1 Estimation of the Number of Author Index Entries Required for One Million Titles 181
6.10.2 Determination of the Number of Bits Needed in The Major, Minor, and Complete Virtual Hash Address . 184
6.10.3 Structure of the Author Hash Index File 185
6.10.4 Number of Content Entries Needed In each Bucket 185
6.10.5 Bucket Size 186
6.10.6 Size of the Author Hash Index File 186
6.10.7 Expected Time to Access a Bucket. 186
6.11 The Coding Scheme for Author Words and Title Words 187
6.11.1 Introduction 187
6.11.2 Measures of the Efficiency of a Coding Technique 187
6.11.3 Description of the Coding Scheme. 189
6.12 The Dictionary Files 191
6.12.1 Introduction 191
6.12.2 Structure of the Pointers to the Dictionary Files 192
6.13 The Dictionary for Title words 194

Page

6.13.1 Estimated Size 194
6.13.2 Expected Decoding Time 196
6.13.3 The Expected Compression of the Title Fields of the Compressed Catalog and the Expected Saving in the File Storage Costs . . . 203
6.14 The Dictionary for Author Words 204
6.14.1 Estimated Sizes 206
6.14.2 Expected Decoding Time for Author Words 206
6.14.3 The Expected Compression of the Author Fields of the Compressed Catalog 206
6.15 The Compressed Catalog File 206

CHAPTER VII PROPOSED FILE STRUCTURE FOR THE REAL-TIME CIRCULATION SUBSYSTEM

7.1 Factors that Influence the Design of the File Structure 211
7.1.1 The Characteristics of the Library for Which the System is Designed 211
7.1.2 The Characteristics of the Circulation Subsystem 211
7.1.3 The Desirable Global Characteristics of the File Structure . 214
7.2 Description of the Files 216
7.2.1 Overview 216
7.2.2 The LC Call Number File 218
7.2.3 The Accession File 228
7.2.4 The Reservations File 235
7.2.5 The Loans File 238
7.2.6 The Patron File 243
7.2.7 The Master Patron File 245
7.2.8 The Accession Directory File . . 252

CHAPTER VIII CONCLUSION 268

BIBLIOGRAPHY 273

APPENDICES 294

APPENDIX I: ALPHABETICAL INDEX TO THE COMMANDS OF THE IT SYSTEM 295

APPENDIX II: SAMPLE TERMINAL SESSION 296

1. Alphabetical Index to the Commands that are Demonstrated 296
2. The Sample Session 297

Page

APPENDIX III: ALPHABETICAL INDEX TO THE FILES OF THE IT SYSTEM 311

APPENDIX IV: THE PROGRAMS OF THE IT SYSTEM 312

1. Alphabetical Index to the FORTRAN IV Programs 312
2. Alphabetical Index to the Entry Points in the FORTRAN Programs . . 313
3. Alphabetical Index to the MTS Command Packages 315
4. The Listings of the FORTRAN IV Programs 316
5. The Listings of the MTS Command Packages 398

LIST OF TABLES

Table		Page
3.4.1	The On-Line Commands of the IT System	51
5.1.1	FORTRAN IV Programs	118
5.1.2	MTS Command Packages	119
6.4.1	Values of c and c/b for Selected Values of b Subject to the Conditions that $P_i \leq 0.001$ and $\alpha = 1$	149
6.13.2.1	Comparison of Six Different Core-Disk Partitions of the Title Word Dictionary . .	199
7.2.3.1	Comparison of Blocking Factors for 12-Byte Logical Records	234
7.2.4.1	Comparison of Blocking Factors for 9-Byte Logical Records	239
7.2.8.1	Comparison of Blocking Factors for 4-Byte Logical Records	255

LIST OF FIGURES

Figure		Page
1.2.1	Classes of Computer-Based Library Automation Systems	5
3.5.1	Examples of Valid Personal User Names and Complete Personal Author Names	54
3.5.2	Examples of Valid Corporate User Names and Corporate Author Names	54
3.5.3	Examples of Valid Author Names	54
3.9.1	Entry of a Command Without Prompting	68
3.9.2	Entry of a Command With Prompting	68
4.1.1	The Logical Disk Files Operated on by the IT System	70
4.1.2	The Primary Files of the On-Line Subsystems	71
4.2.1	The Files AUTHORS, LOANS, and CATALOG	73
4.2.1.1	Sample Entries from the File CATALOG	77
4.2.2.1	Sample Entry from the File LOANS	83
4.2.3.1	Sample Entry from the File AUTHORS	83
4.3.1	The File USERS and Related Files	85
4.3.1.1	Sample Entry from the File USERS	91
4.3.2.1	Sample Entries from the File USERINDEX	91
4.5.1	General Flow Chart for the Modules BATCHUPDATE and PICKUP	98
4.5.1.1	Sample Entries from the File NEWBOOKS	103
4.5.4.1	Sample Entry from the File DELETIONS	107
4.6.1	Off-Line Enrollment of New Users with the Batch Mode Module BATCHENROLL	108
4.6.2	Enrollment of New Users from the Terminal	110
4.6.3	Sample Entry from the File NEWUSERS	115

Figure Page

5.2.1 The General Structure of the On-Line Module LIBRARY 121

6.1 Conceptual View of an Integrated, On-Line, User-Oriented, Library System 126

6.1.3.1 The Basic Components and Structure of the File Organization for the On-Line Catalog 131

6.1.3.2 The Structure of the File Organization for the On-Line Catalog when a Thesaurus is Included 132

6.1.3.3 The Relationships Among the LC Call Number Hash Index File, the LC Call Number File, the Compressed Catalog File, and the Files of the Real-Time Circulation Subsystem . . . 134

6.3.1 Basic Structure of the Hash Index File: Overview 144

6.3.2 Basic Structure of the Hash Index File: Details of Buckets 145

6.6.1.1 Searching the Hash Index Files 168

6.12.2.1 The Relationship of the (K,J,I) Codes to the Structure of the Dictionary Files 193

6.12.2.2 The Structure of the Three-Byte Dictionary Pointer Field 195

7.2.1.1 The Files of the Real-Time Circulation Subsystem and Their Relationships with Each Other and the Compressed Catalog File . 217

7.2.2.1 Structure of Records for Monographs in the LC Call Number File 220

7.2.2.2 Structure of Records for Closed Multi-Volume Sets in the LC Call Number File 221

7.2.2.3 Structure of Records for Open Multi-Volume Sets in the LC Call Number File 222

7.2.2.4 Structure of LC File Records for Both Bound and Unbound Periodicals 223

7.2.2.5 Structure of Pointers to the LC Call Number File 226

Figure Page

7.2.2.6 An Alternative Structure for the Physical Records of the LC Call Number File 227

7.2.3.1 Structure of Entries in the Accession File . . 232

7.2.4.1 Structure of Records in the Reservations File. 236

7.2.4.2 The CORAL Ring Structure that is Formed by Each Patron's List of Reservations 237

7.2.5.1 Structure of Records in the Loans File 241

7.2.5.2 The CORAL Ring Structure that is Formed by Each Patron's List of Borrowed Books 242

7.2.6.1 Structure of Records in the Patron File . . . 244

7.2.7.1 Structure of Records in the Master Patron File 247

7.2.7.2 Structure of Physical Records in Master Patron File When the Logical Records are Blocked 16 per Variable-Length Physical Record 248

7.2.7.3 Structure of Pointers to the Master Patron File When the Scheme Illustrated in Figure 7.2.7.2 is Used 249

7.2.7.4 Mapping the Variable-Length Master Patron Records into Fixed Length Physical Records . . 251

7.2.8.1 Structure of Accession Directory Records: Case One 253

7.2.8.2 Structure of Accession Directory Records: Case Two 253

7.2.8.3 Linkage Setup Between Accession File and LC File for the Case of Monographs with No Added Copies 257

7.2.8.4 Linkage Setup Between Accession File, Accession Directory File, and LC File for the Case of Monographs with Added Copies . . 258

7.2.8.5. The Pooling Scheme for Vacated Accession Directory Entries 260

7.2.8.6 Linkage Setup Between Accession File, Accession Director, and LC File for the Cases of Open and Closed Multi-Volume Sets . 262

Figure Page

7.2.8.7 Linkage Setup Between the Accession File and the LC File for the Special Case of a Closed Multi-Volume Set with the Same Number of Copies in Each Volume and No Added Copies 264

7.2.8.8 Linkage Setup Between Accession File, Accession Directory, and LC File for the Case of Journals 266

CHAPTER I

INTRODUCTION

1.1 The Need for Automated Library Systems

During the past 25 years, many libraries, particularly academic libraries, have had to cope with several so-called "explosions" which have placed severe demands on their resources. These explosions include the information explosion, the publication explosion, the user population explosion, and the wage/price explosion.

The information explosion has been caused by the rapid growth of knowledge and also by the breakdown of barriers between disciplines that were once regarded as more or less mutually exclusive domains. It has also been caused by the birth of many new disciplines that overlap several older ones. Examples of such new disciplines are computing science, information science, biochemistry, biophysics, and medical engineering.

One result of the information explosion is that the elegantly constructed classification schemes traditionally used in libraries have become very cumbersome for both the librarian and the patron.

A consequence of the publication explosion is that libraries must now acquire, process, store, and circulate ever-increasing amounts of information-bearing material.

The user population explosion has been particularly

evident in the instance of academic libraries. Greatly expanded university enrolments at both the undergraduate and graduate levels have required these libraries to acquire, circulate, and tightly control greatly increased quantities of material relating to any given subject area.

Graduate students, in contrast to undergraduates, place special demands on the library since they make heavy use not only of the library's shelf collection of monographs, but also of journals, abstract journals, and specialized services such as interlibrary loan and abstracting services.

Additionally, a library collection that is more than adequate for the needs of undergraduate students often fails to satisfy the needs of graduate students in terms of depth, breadth, and timeliness. Graduate students and others engaged in research are particularly intolerant of long delays in the acquisition of new publications that relate to rapidly developing disciplines such as computing science.

The wage/price explosion has caused difficulties in the acquisition of additional personnel, both clerical and professional, to help cope with the greater demands generated by the first three explosions.

With the passage of time it became clear that extra personnel, when available, failed to solve the ever-growing problems associated with the various "explosions."

Many librarians realized that their problems could not be solved merely by quantitative changes in old procedures, but that qualitative changes and new methods were required. Among the most promising of the new methods was automation, particularly automation dependent on the digital computer. The application of computers to library functions held the

promise of greater throughput, at higher efficiency with less personnel and at reduced unit costs.

The manner in which library automation systems have evolved to meet the needs mentioned above is described in the sections that follow. A particular on-line library automation system, called the IT system, has been developed at the University of Alberta and a full discussion of it forms the main content of the present thesis.

1.2 Evolution of Single Task and Integrated Systems

Library automation, defined as the application of computers or related data processing equipment to library operations, was perhaps first introduced in 1936 when R. H. Parker applied punched card accounting equipment to perform circulation functions at the University of Texas [1]. Parker subsequently devised punched card routines for serials control and acquisitions.

The work of Parker appears to have been largely ignored by librarians until the late 1950's and early 1960's. During that period many librarians designed and installed serials control, accounting, circulation, ordering, and catalog production systems based on unit record equipment [2, 3, 4, 5].

While most of the librarians engaged in automation activities concentrated on the development of mechanized procedures based on unit record equipment, there were some who explored potential uses of the electronic digital computer. The first reports of operational computer-based library

automation systems appeared in the literature in 1963.

Among such reports was that of H. L. Griffin [5] that dealt with the conversion of book ordering, catalog production, and circulation procedures from an IBM 407 punched card accounting machine to an IBM 1401 card processor and later to an IBM 1401 card/tape processor. M. Griffin [3] described the use of an IBM 1401 to produce author and title sequence book catalogs.

Since 1963 the trend in the development of computer-based library automation systems has been toward either the single task system or the total, or integrated, system. Each of these systems may be developed into either an on-line or an off-line system (See Figure 1.2.1).

Single task systems are characterized by two properties. The first is that they are designed to automate procedures in one isolated area of library activity where a specific and critical need exists. The single activity might be, for example, acquisition, cataloging, serials control, information retrieval, or circulation. The second property is that they can be implemented without attention to integration into an overall automation system for the entire library.

Total systems, in contrast to single task systems, are characterized by the property that they treat the activities of the library as a unified whole. They may, of course, consist of a set of interrelated subsystems that, when all are implemented, accomplish the automation of the entire

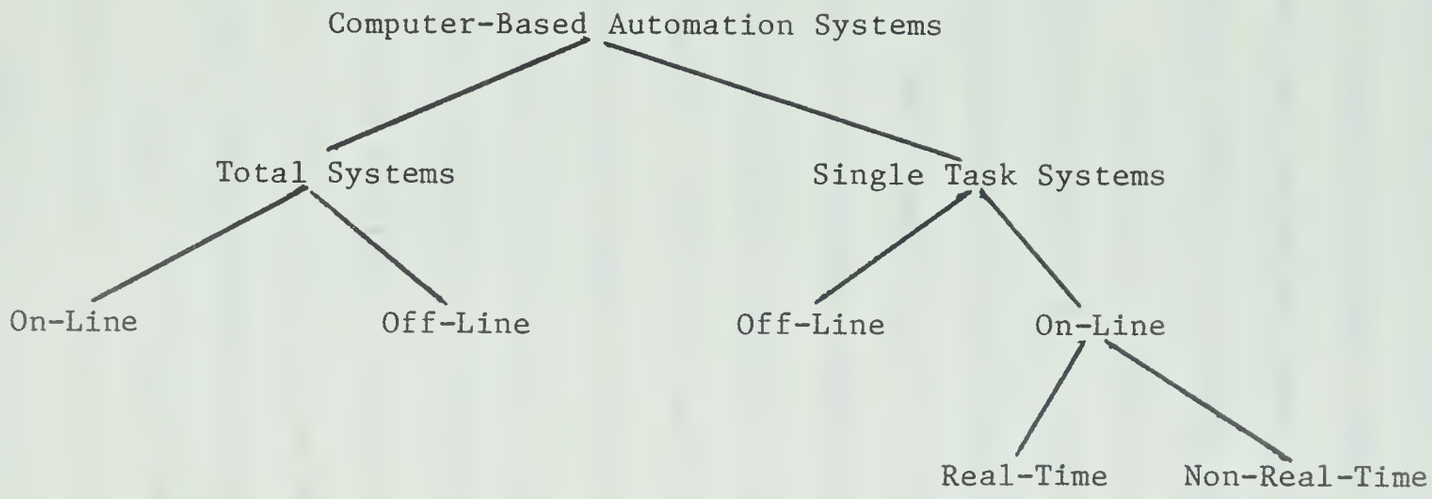


Figure 1.2.1 Classes of Computer-Based Library Automation Systems

library. They are further characterized by the use of a central data bank stored on tape, disk, or data cell, by the minimization of input operations during technical processing, and by the repeated use of information for many different purposes once it has been entered into the central data bank.

In a total system, when a request for a new book or document is first processed all the available bibliographic and processing information is entered into the central data bank. The resulting initial, or basic, record has its data elements corrected, added to, or deleted, as the requested book or document passes through the ordering, receiving, cataloging, shelving, and circulation stages. In addition to this basic record within the central data bank being used to produce the original order, it is also used to produce routing information, bibliographic entries for book, card, or on-line catalogs, spine and pocket labels for books, book cards for the circulation system, and reports to the requestor and the library administration.

Total, or integrated, systems have been discussed by numerous authors including D. N. Kraft [6], D. M. Heaps and H. S. Heaps [7], M. A. Jennings [8], F. G. Kilgour [9], I. A. Warheit [10], S. Alanen et al. [11], R. M. Hayes [12], L. A. Schultheiss et al. [13], R. H. Parker [14], M. Griffen [15], and C. T. Payne [16].

1.3 Off-Line Single Task Systems

Off-line systems are oriented towards the use of punched cards, magnetic or paper tape, and printed listings.

In such systems transactions are collected on cards, or on magnetic or paper tape, over fixed periods of time. The resulting batches of transactions are then processed to either update the master data files, which are usually stored on magnetic tape, or to generate printed listings of such items as books on-order, books received, books on-loan, and books overdue. These listings are then used to answer queries asked by the librarian or the patron.

On-line systems, in contrast to off-line systems, are oriented towards the use of typewriter, teletype, or CRT terminals, magnetic disks, and data cells. For the purposes of this discussion, on-line systems are amended to include only those systems that allow interrogation of the master disk or data cell files from terminals. They specifically exclude those systems that use terminals only as data collection devices. This means that circulation systems based on the IBM 357 Data Collection System or on the IBM 1030 Data Collection System are considered to be off-line systems, unless terminals are used to interrogate the disk files.

On-line or terminal-oriented systems are further subdivided into real-time and non-real-time systems. In real-time systems the master disk files are updated as soon as transactions are entered from the terminals. In non-real-time systems transactions entered from the terminals are collected on card, tape, or disk files until a certain number have accumulated or until a certain period of time has elapsed;

the batch of transactions is then processed to update the master disk files.

Of the four main branches of computer-based library automation systems that include off-line single task systems, on-line single task systems, off-line total systems, and on-line total systems, the off-line single task systems were the first to develop, primarily because of the relative ease and low cost with which they can be implemented. Some of the first operational examples were reported in 1963 [3, 5].

The tasks most frequently chosen for automation through off-line single task systems are listed in i) to iv) below:

i) Production of book-form catalogs [3, 5, 18, 19, 20, 21, 22, 23].

ii) Serials control [5, 24, 25, 26, 27, 28].

iii) Circulation [5, 17, 29, 30, 31, 32, 33].

iv) Book order and receipt [5, 24, 32, 34, 35, 36, 37, 38].

When applied to circulation many of the off-line systems generally fulfilled their promised functions such as the following:

i) Reduced clerical work for the library staff. The tedious tasks of sorting, tagging, posting, slipping, checking, and filing of loan records and the preparation of reserve, recall, availability, hold, due, overdue, and fines notices had been transferred to the computer system.

ii) Reduced patron effort. The tedious chore of filling out charge slips had been eliminated. Usually a book

card and a reader card or badge were read simultaneously by an input station of an IBM 357 Data Collection System. The information on these cards was then transmitted to a centrally located card punching station which produced one or two transaction cards.

iii) Ease of production of lists of material on loan to given borrowers as well as lists of borrowed material arranged by author and title or by call number.

iv) Ease of production of accurate statistical reports.

However, these off-line systems still possessed certain disadvantages some of which are listed below:

i) The handling of bulky loan-record files had been replaced by the handling of bulky computer listings, such as the listings of the loan file in patron name or identification number, author and title, and call number sequences.

ii) Listings were often at least twenty-four hours out-of-date.

iii) Errors could not be detected at the time transactions were made, but only when the transaction cards were processed by the computer many hours later.

iv) Since the computer centre was often not in the same building as the library, the cards, tapes, and listings had to be transported back and forth with attendant risks of being lost, damaged, or delayed.

v) The off-line circulation system could not stop

delinquent patrons from borrowing additional books.

vi) Discharging returned books required a manual search of a "holds" file or listing and a twenty-four hour, or longer, delay in the shelving of the returned books to ensure that the current "holds" listing included reservations placed the same day that the books were returned.

vii) Whenever a book was renewed the librarian had to manually search a reservations file to determine whether or not a reservation had been placed for the book.

The off-line single task systems that handled tasks other than circulation fared somewhat better, but still shared many of these faults that led librarians to consider the use of on-line systems.

1.4 On-Line Single Task Systems

About 1964 librarians started to give serious thought to the development of on-line automation systems, and soon the first on-line single task systems became operational. Thus B. B. Bateman and E. H. Ferris [39] reported an operational acquisition system in 1965. Aside from this on-line acquisition system, an on-line acquisition system at the Oregon State University Library [40], an on-line serials system at Laval University Library [41], and a large number of experimental on-line information retrieval systems found outside the library environment, most on-line single task systems have been circulation systems.

The reasons for the relative popularity of on-line circulation systems are not difficult to find. Firstly, the off-line circulation systems mentioned in the previous section had largely fulfilled their designers' aims. Secondly, most of their inherent disadvantages can be eliminated in on-line, non-real-time systems, all can be eliminated in real-time systems. Additionally, on-line systems provide a centralized file of loan records. This is a distinct advantage to libraries with several branches.

Real-time systems provide absolutely up-to-date information since there is no delay in processing loan, return, recall, or reserve transactions. The circulation functions are readily isolated from other library functions. The circulation functions are relatively easy to analyze and program. Furthermore, if the library is already operating an off-line circulation system based on the IBM 357 or IBM 1030 Data Collection System then the existing book and reader cards, and perhaps the data collection system, may be used with an on-line circulation system.

If the library has not been operating an automated circulation system then book and reader cards probably do not exist; but this need not be an obstacle if the library adopts a system that uses typewriter terminals for input of transactions. Such systems have been proposed by R. T. Kimber [42, 43], G. J. Lazorick and J. P. Herling [44], R. H. Stanwood [45], and D. M. Heaps and H. S. Heaps [7].

Circulation files do not normally require massive

quantities of disk or data cell storage. This is in contrast to the files required, for example, for an on-line catalog searching system. Also, a small to medium size computer is able to handle the expected processing loads.

These considerations, which have favoured the development of on-line circulation systems, have been discussed in more detail by R. T. Kimber [42, 43, 46], C. R. Umstead and F. E. Croxton, [47], EDP Systems Development Services [48], C. J. Lazorick and J. P. Herling [44], R. A. Kennedy [49], and C. J. Surace [50].

Seven on-line circulation systems, three of which are real-time, have been reported as operational in libraries. The first of these was a non-real-time system that began operations at the Illinois State Library in 1966 [51]. R. T. Kimber [52] reported that the West Sussex County Library was the first British library to have on-line interrogation of its circulation files, but this library's system is definitely not a real-time system as the master loans file is updated in batch mode once each week. Simon Fraser University [53] has an operational non-real-time circulation system based on an IBM 1030 Data Collection System, an IBM 360/50 CPU with 512K of core memory, and IBM 2260 display terminals.

The first genuine real-time circulation system was the BELLREL system reported by R. A. Kennedy [49], which became operational in March 1968 and services three Bell Laboratories' libraries in New Jersey. Work at The Queen's University of Belfast, which has produced a second operational real-time

system, has been reported by R. T. Kimber [42, 43] and by A. H. Boyd and E. J. Walden [54]. More recently, a third real-time circulation system became operational at Ohio State University [45]. An eighth on-line circulation system, a real-time one, was reported by G. J. Lazorick and J. P. Herling [44] to be under development at the University of Buffalo.

1.5 Off-Line Total Systems

Developmental work on total systems began about the same time as for single task systems. In 1962 L. A. Schultheiss, D. J. Culbertson, and E. M. Heiliger [13] published the results of a library systems study done between 1960 and 1962 at the University of Illinois. Their systems study recommended that the university library design and implement a computer-based, off-line, total system, but did not define the details of the design. R. D. Kozlow [24] reported that a project to carry out the detailed design and implementation of this proposed total system began in January 1963 and continued until July 1965 at which time it was dropped in favour of a new project that would carry out the design and implementation of an on-line total system. Of the four subsystems of the proposed off-line total system, comprising acquisitions, serials, catalog printing, and circulation, only the acquisitions and serials subsystems were programmed and tested before the project was terminated.

Meanwhile, in September 1963, E. M. Heiliger moved to Florida Atlantic University and started the design and implementation of an off-line total system based on the

recommendations of the previously mentioned library systems study done at the University of Illinois.

In 1965 Heiliger [55] reported that the library at Florida Atlantic University had implemented the book catalog printing subsystem, the serials subsystem, and the off-line circulation subsystem; and was programming the acquisitions subsystem. Since that time several other implementations of off-line total systems have been reported [56, 57, 58, 59, 60, 61, 62, 63].

As in the case of the off-line single task systems discussed earlier, off-line total systems have provided their implementers with many benefits, most of which have been outlined by Hayes International Corporation [56] in their dicussion of the ALPHA-1 system in use at the Redstone Scientific Information Center. However, such systems are considered to be several steps closer to the ideal library automation system.

1.6 On-Line Total Systems

Some of the advantages of on-line systems over off-line systems have been described already. Real-time total systems offer further advantages which are summarized below:

i) Terminal-oriented systems provide direct communication with the computer and, therefore, help to eliminate the preparation, handling, and storage of forms, cards, and tapes.

ii) Real-time, interactive operation provides that errors in input transactions may be detected and corrected

immediately, rather than after a delay that may be several days in length.

iii) A real-time library automation system can instantly provide absolutely up-to-date information about all books or documents requested, ordered, received, outstanding, in cataloging, in the bindery, on loan, reserved, recalled, due, overdue, and so forth.

iv) The printing and handling of voluminous computer listings is greatly reduced in an on-line system since the librarian or the patron can directly interrogate the computer system's disk or data cell files to obtain accurate and timely information.

v) In a real-time system, changes, additions, or deletions may be carried out immediately without the necessity of waiting until the next batch processing time.

vi) An on-line system allows a reduction in the amount of material that must be transported from place to place. There is little need to move forms, cards, and tapes between the library and the data processing center. In addition, there is no need to distribute listings and other products such as book cards, spine and pocket labels, book orders, routing slips, reports, and the like as these products of the computer system can be provided through terminals, at the places where they are needed and at the times they are needed. The terminals may, of course, include high-

speed printers and card punches.

vii) With an on-line system, the centrally stored disk or data cell files, which contain the data base of bibliographic and in-process information, can be made accessible anywhere through terminals linked to the computer via standard telephone lines. Therefore, an on-line system can service several geographically dispersed libraries as easily as it can serve a single library. It thus provides opportunities for the pooling of holdings, the centralizing of technical processing activities such as ordering, receiving, and cataloging, and the sharing among several libraries of the cost of a computer system large enough to handle the requirements imposed by an on-line total system.

Examples of systems that illustrate the features outlined in (vii) include the BELLREL real-time circulation system [49], IBM's multi-library on-line acquisition system [39], the proposed on-line network for five Washington, D.C. libraries [64], and the planned Ohio College Library Center Regional Library System [65, 66]. The latter system is a multi-library, on-line, total system that may eventually include shared acquisitions and cataloging, a union catalog of holdings accessible through remote terminals, a circulation system, a serials system, and an information retrieval system. An additional example is provided by NELINET, New England Library Information Network, which is reported [67]

to be developing an on-line, time-shared system to provide college libraries in New England with centralized technical processing services and with on-line interrogation, via display terminals, of the central data bank in support of acquisition, cataloging, and reference activities.

Further discussion of the advantages of on-line total systems may be found in the work of R. M. Hayes [12], D. P. Hammer [68], M. A. Jennings [8], A. N. Grosch [69], F. G. Kilgour [65], I. A. Warheit [10, 70], W. R. Negent [67], and C. J. Surace [50].

The on-line total system started to receive serious attention during 1965 and 1966 when developmental work was reported at the Redstone Scientific Information Center [56, 57] and at IBM's Los Gatos Library [12]. In 1966 the University of Chicago [16, 71, 72] and in 1967 Washington State University [73] started work on on-line systems that are best described as on-line, integrated, technical processing systems of the type envisioned by R. M. Hayes [12], M. A. Jennings [8], and I. A. Warheit [10] because neither appears to involve on-line catalog searching. In 1968 T. Burgess et al. [73] reported that an on-line acquisitions system, LOLA, was operational at Washington State University.

The previously mentioned on-line library network systems under development in Ohio and New England are both extensions of the on-line, integrated, technical processing system from one to many libraries. Both of these systems may eventually be extended in scope to cover on-line catalog

searching.

The concept of an on-line library automation system involving on-line searching of the library's catalog was outlined by D. R. Swanson [74] in 1963 and it has since been developed further by several authors including F. G. Kilgour [75, 76] and H. S. Heaps and D. M. Heaps [7]. During 1967 and 1968 projects aimed at the design and implementation of such a system were started at Stanford University [77], the University of California at Berkeley [78], the University of Alberta [7], and the University of Utah [79].

While comparatively little work has been done on the on-line search of library catalogs, a great deal of related experimental work in information retrieval has been undertaken outside the library environment. It has included work relating to on-line information retrieval using small data bases consisting of, at most, a few thousand document surrogates.

Some on-line information retrieval experiments that have been conducted since 1963 are TIP at MIT [80, 81, 82, 83], BOLD at Systems Development Corporation [84], CONVERSE [85] and DIALOG [86] at Lockheed, INTREX at MIT [87, 88, 89, 90, 91, 92, 93, 94], Automation of a U.D.C. Based Library for Searching Purposes [95], LC/MARC on MOLDS at Syracuse University [96], an On-Line Personal Documentation System at the University of Alberta [97], AUDACIOUS [98], GRINS at Lehigh University [99], an On-Line Information Retrieval System with Application to Canadian History [100], LEADER at

Lehigh University [101], Time Sharing Chemical Information Retrieval System at the University of Pennsylvania [102, 103], REQUEST at the University of Illinois [104], TIRP at MIT [105], and EASY ENGLISH at the University of Pennsylvania [106, 107, 108]. Of all these experimental information retrieval systems, the one which is most relevant to on-line search of a large library catalog is Project INTREX.

One off-line experimental information retrieval system should also be mentioned because of the important work that has been done with it on query formulation, search strategies, dictionaries, and thesauri. It is the SMART system of G. Salton at Harvard and Cornell Universities [109].

1.7 Statement of the Problem

The foregoing sections have given a brief outline of the development of library automation systems from simple punched card systems to on-line total systems. The design considerations of this latter type of system form the main subject of the present study.

The work reported in this treatment is concerned with two problems. The first problem is the design, implementation, test, and evaluation of real-time integrated library automation system to handle the user-oriented functions of a small specialized library. The second problem involves the design of a file structure and the related file access strategies for a real-time integrated library automation system to serve the needs of a large university library. Central to such a system is an on-line catalog to be used by both patrons and staff.

The central problem of file design is, however, embedded in a more general problem: the overall design of the automation system. The overall design problem may be subdivided into seven aspects or subproblems as follows:

i) Design criteria. This is in turn composed of three aspects:

   a) Systems analysis of a large university library in terms of its subsystems, the functions of those subsystems, the information required to perform those functions, the information and materials processed by the subsystems, and the required files and their expected sizes.

   b) General specifications for the library automation system in terms of the functions to be performed, the mode of performing each of these functions, either on-line or off-line, the access points required, the desired response times for on-line functions, the types of equipment to be used, and the desirable characteristics of the overall system.

   c) Design contraints imposed by the nature of the available hardware, including such characteristics as the expected access times, the minimum and maximum access times, the capacities, and the data transmission rates of mass storage devices such as drums, disks, and data cells.

ii) Command language. This forms the basis of the user-

system interface and brings up such considerations as the particular commands to be implemented and the desirable characteristics of the command language.

iii) Record content and structure. This involves such considerations as the desirable characteristics of the record structures, and the structuring of the records to achieve those desirable characteristics.

iv) File organization. This involves such considerations as the possible basic file organizations and their advantages and disadvantages, organization of the system's files in order to obtain an optimal trade-off between file storage cost, updating time and cost, and file search times. The latter times determine the query response times.

v) Data compression and coding. The main bibliographic file of an automation system for a large university library is expected to be very large, of the order of 300 to 500 million characters. Hence, it is quite costly to store this file on-line on disks or data cells. The question arises as to whether or not it is feasible to use some technique to compress the bibliographic data.

vi) Multi-terminal support. This involves the question as to whether the automation system should time-share among requests from different terminals, or whether it should service to completion each request in turn. If the system is to time-share, or otherwise overlap

activity among requests, then there arises the problem of arranging synchronization of activities, particularly file accessing, to maximize the amount of overlap in the processing of different requests. Problems of file security and request interlock must be solved. Decisions must be made regarding whether requests from different terminals should be batched and, if so, how this should be done.

vii) Equipment required to implement the design. There arises the question of whether the library automation system should be implemented on a dedicated computer system or whether it should be subsumed under a general-purpose time-sharing system which serves many other users besides the library. If it is feasible to implement the library automation system under a general-purpose time-sharing system, then it is of interest to determine which time-sharing system, for example, CP/CMS, MTS, or TSS/360, is most suitable. If the system is to be implemented on a dedicated computer system it is important to determine the minimum hardware requirements in terms of such items as CPU, core memory, drums, disks, and data cells. The types of terminals to be used must be determined. The expected cost of the automation system is also an important factor to be determined.

Of these seven aspects of the overall design problem, the third, fourth, and fifth are emphasized in this treatment.

In particular, a new method of storing and searching the inverted indexes of an on-line catalog system is proposed and analyzed.

The structure of the remainder of this treatment is as follows:

i) Chapter II is a detailed review of some of the more important work that has been done towards the implementation of on-line total library automation systems.

ii) Chapters III, IV, and V describe the IT system, which is a real-time integrated library automation system that handles the user-oriented functions of a small specialized library.

iii) Chapters VI and VII deal with the design of a file structure for a real-time integrated library automation system that handles the user-oriented functions of a large academic library.

iv) Chapter VIII is an evaluation of the work reported in the first seven chapters.

CHAPTER II

REVIEW OF THE SALIENT LITERATURE RELATING TO ON-LINE LIBRARY AUTOMATION SYSTEMS

2.1 Introduction

It is outside the scope of this study to discuss in detail the literature pertaining to all of the on-line systems mentioned in Chapter I. Therefore, this review covers only those on-line systems that are considered to be closely related to the type of library automation system under discussion in later chapters of this treatment and to be representative of the state-of-the-art as reported in the literature.

2.2 The LOLA Subsystem

2.2.1 Overview of LOLA

One of the most advanced on-line acquisitions systems reported in the literature is the LOLA, Library On-Line Acquisitions, Subsystem, which has been in operation since April 1968 at the Washington State University Library [73].

LOLA is the first operational module of a planned integrated technical processing system that, when fully implemented, is to encompass all activities concerned with the ordering and processing of books and documents before they appear on the library shelves. These technical processing activities include preparing and verifying

purchase requests, ordering, receiving, invoice-handling, accounting, accessioning, binding, subject and descriptive cataloging, preparing and affixing book spine and book pocket labels, preparing book cards for an IBM 357 circulation system, producing standard "3 by 5" catalog cards, and shelving the processed books and documents.

In their report on LOLA, T. Burgess and L. Ames [73] also mentioned a planned circulation system that would allow on-line inquiry and modification of the circulation file, but they did not give even a general description of this circulation system. Their report contained no indications that they were either implementing or planning to implenent an on-line catalog system.

2.2.2 LOLA's Disk Files

The entire LOLA Subsystem centers on a disk file that is accessible through on-line terminals for inquiry or modification at any time [73]. This primary technical processing file, called the In-Processing File, contains the record of every book from the time the order for it is initiated until the time the catalog card for it is filed in the library's card catalog and the book is placed on the shelf.

The In-Processing File contains five basic groups of information. It contains status information, which indicates the current state of an ordered item and the operations that have been completed. It contains budget information and other information necessary for the purchase of books or documents. It contains records of transactions

that have taken place, such as the entry of invoice information or the disbursement of funds. It contains ordering information, which includes the vendor's name and number, and instructions to the vendor. Finally, it contains bibliographic information.

2.2.3 Data Capture and Flow in the Integrated System

Whenever an order for a book or document is initiated, all the bibliographic information that is known about the book is entered into the system's files [73]. Thus, when the cataloging subsystem has been implemented, the catalogers will not need to recopy or re-enter all the bibliographic information required to perform the cataloging function. Instead, they will use editing facilities provided through display terminals to modify the bibliographic information that already exists in the In-Processing File. The modified bibliographic record will then be used to produce standard "3 by 5" catalog cards.

This approach contrasts sharply with that taken in the usual manual or semi-automated system in which there are many recopies and many reverifications of the same bibliographic information as a purchase request progresses from the initial searching phase to the final card production phase.

2.2.4 Search of the In-Process File

On-line search of the In-Process File in the current version of LOLA is limited to one access point, the purchase order number. Burgess and Ames plan to add the ability to

search this file on-line from the title, author, and vendor approaches, but in the interim their system provides two printed indexes to the records in this file.

One index is arranged alphabetically by title; the other is arranged numerically by purchase order number. These indexes provide the librarian with the following information about a requested book: the purchase order number; the author and title; the volume, issue, or edition; the date ordered, if the book has been ordered; the date received, if the book has been received; the accession number assigned to the book; and the final disposition of the book. These listings are used primarily by the Searching Department to determine whether or not a requested book is already on-order.

To supplement these indexes, all information associated with current orders is reproduced bi-weekly in the form of a very large and costly listing that is alphabetized by title.

Adequate provision for on-line search of the In-Process File would largely eliminate any need for these listings.

In addition to these listings of the In-Process File, LOLA provides status and accounting reports. The status reports are produced on demand and give the purchase order numbers of those purchase orders that are in a specified state. The accounting reports are produced automatically and include reports that indicate allocations and expenditures for each division of the library as well as comprehensive reports for the library administration and university comptroller.

2.2.5 File Organization and Record Structures

T. Burgess and L. Ames [73] have neglected to give any details of the file organization or the internal record structures employed by LOLA. In addition, the relationship between the In-Process File and the Orders-Out File, both of which are on-line, is ambiguous in their report.

2.2.6 Evaluation

The integrated technical processing system of which LOLA is the first operational module falls short of fully exploiting the capabilities of an on-line computer system. Evidence of this is provided by the heavy reliance on batch processing to produce listings of the various files and to produce accounting cards which are later processed by a separate set of batch programs, and by the poor facilities provided for on-line searching of the In-Process File. It further appears that this system has been designed to fit into the "Procrustean Bed" of a fragmented library organization. Ideally, the overall organization of the library should be modified so as to aid in the full exploitation of the potentialities of an advanced library automation system. Finally, it should be possible to carry the design two steps further and integrate an on-line catalog system and a real-time circulation system into an overall on-line system that would aid the performance of all of the library's functions. The machine-readable data base for these two additional systems already exists in the In-Process File. There may, of course, be a problem of converting the existing manual card

catalog to machine-readable form.

2.3 The BELLREL System

2.3.1 Introduction

The current state-of-the-art in on-line circulation systems is probably represented by the BELLREL System [49] and by the Ohio State University Libraries' System [45]. The planned system at the University of Buffalo [44] appears to have faded from the scene.

The real-time system reported by A. H. Boyd and E. J. Walden [54] is limited to handling charge, discharge, and simple queries, and it appears to involve but a single on-line console. As in the Simon Fraser system [53], only records pertaining to books currently charged out are permanently stored in the on-line disk file. Boyd and Walden's lucid description of this system includes much more detail about procedures, commands, record structures, file organization, and programming than is commonly supplied in descriptions of other on-line library automation systems.

2.3.2 Overview of BELLREL

The BELLREL System [49] is a real-time circulation system which services three Bell Laboratories' libraries located in New Jersey. Each library in this network has two IBM 1050 terminals connected via dial-up telephone lines to an IBM 360/40, which has its 256k of core memory partitioned to allow the real-time circulation system to run in the "foreground" while routine batch operations for other users of the

computer are performed in the "background." The IBM 1050 terminals incorporate keyboard, printer, and card reader facilities for maximum flexibility in handling all types of transactions and queries.

2.3.3 File Organization and Record Structures

The disk records that underlie the circulation system are of two main types, patron records and publication records [49]. They are stored on a single IBM 2316 disk pack.

The patron records cover about 19,000 people who are mostly employees of Bell or its sub-contractors. Each patron record is 161 characters long and contains the patron's five-digit payroll account number, name, department number, telephone number, location, and occupational class. It also includes space for three book loans and for pointers to overflow loan trailers stored elsewhere on disk. The patron file is organized by the payroll account number which is typed in, or read from an identification card, for all transactions, such as loans or reservations, that require it. These patron records are accessed by the IBM Index Sequential Access Method.

The publication records vary in length, format, and method of access depending upon the class of publication. The circulation system currently encompasses five classes of publications. These are books, journals, trade catalogs, college catalogs, and continuations and multiple-volume titles cataloged as sets. Each title in each class is assigned a unique six-digit number, the first digit of which identifies the class. Particular copies of a given title are

differentiated by a two-letter code that indicates the holding library and by a two-digit copy number that indicates the particular copy at the holding library. The complete ten-character identification number is incorporated into pre-punched item identification cards and into item labels. These item cards and labels also contain the author, title, and call number of the book. The publication file is, essentially, a union shelf list of the holdings of all three participating libraries.

The disk record for each monograph title is 188 characters long and contains such information as the book number, 43 characters of author-title, the call number, copies by location, three loan fields, two reserve fields, and pointers to loan and reserve trailers which are stored elsewhere on disk. Each loan field contains the borrower identification number, the due date, the copy number, and the status of the loan. Identification numbers for monographs are assigned sequentially by the computer during update runs. Monograph records are accessed by the direct key-to-unique address form of direct access (See IBM [110] for an explanation of this method.).

Cataloged continuations and multiple-volume titles share a different type of disk record that permits grouping of volumes that belong to sets or to series. The volumes of one title are handled in one 288-character record under the same identification number. Up to 100 additional volumes may be handled in succeeding records. All records of each

set carry the same first five digits in their identification number instead of having different numbers as would be the case if they were handled as monographs. Access to these records is by ISAM.

The principal advantages of this approach, aside from the grouping of sets, are a saving of disk space and the ability to readily obtain use statistics for each set as a whole as well as for the individual volumes in the set [49]. The prime disadvantage is that a transaction that deals with a specific volume in a set must include both the basic access number and the volume number [49].

Records for journals, trade catalogs, and college catalogs are all 155 characters long and share a common format. Each record contains the item identification number, 48 characters of the title, two loan fields, one reserve field, and pointers to loan and reserve trailers. All three types are accessed by ISAM.

Unlike the records for catalogs, monographs, and sets, the journal records are not permanent residents of the publications file, but are kept only as long as they are needed to record a current loan or reservation.

The loan and reserve trailer records for each publication class accommodate overflow. These records vary in quantity and size depending upon function, publication type, and predicted need.

2.3.4 On-Line Transactions

The BELLREL System provides twenty-two command codes

to handle loans, returns, renewals, reservations, and queries in real-time. Each command code consists of two or three characters and is always keyed into the IBM 1050 terminals. R. A. Kennedy [49] included in his report two tables which list the functions and formats of all twenty-two command codes, but only some of the more common commands are discussed here.

The loan command consists of the two-character code LM, keyed into the 1050 terminal, plus an item number and a patron number. Both the item and patron numbers can be either keyed in or read from cards. The loan period is assigned by the computer program on the basis of the first digit of the item number. This first digit, it may be recalled, defines the publication class of the item. If desired, another command can be used to change the automatically assigned loan period.

The return command consists of the two-character code LC, keyed into the terminal, plus an item number which is either keyed in or read from the item card. If a reservation for the returned title exists anywhere in the library network, then the first copy returned is automatically charged to the first person in the reserve queue. The loan period assigned by the computer program depends upon whether or not there is a waiting list for the book.

The reserve command consists of the two-character code LA, the patron number, and the item number, all of which are keyed into the terminal. If all copies of the requested

title are on loan, then the system tells the requester where he stands on the reserve queue for the title. If all copies are not on loan, then the system tells the requester which branch library still has copies available as well as placing him at the head of the reserve queue. One of the branch libraries that still holds a free copy is then instructed by telephone or by terminal-to-terminal communication to mail the item to the requester and run a return transaction against the item and so cause it to be automatically signed out as previously described.

BELLRELL also provides query commands that yield such information as the status of a specific title or copy, the list of items on loan to a specific patron, and the display of a particular patron's complete record.

2.3.5 Batch Processes and Products

Periodic runs of batch programs produce overdue notices, daily loan lists, high-demand lists, zero-loan lists, missing-item lists, and other reports such as circulation statistics by subject class, library, user department, and the like.

2.3.6 Evaluation

BELLREL is reported [49] to be achieving its design objectives which include improved service through computer pooling of dispersed library collections, up-to-date reporting on the status of any publication, immediate identification of all items on loan to a person, automatic follow-up on reserve

queues, reduced clerical labour, better inventory control, and improved feedback for the library management. From the librarian's point of view BELLREL is a remarkable advance, but from the programmer's point of view BELLREL is simply an application of well-known and well-tested techniques.

2.4 The Berkeley Bibliographic File Organization Project

2.4.1 Introduction

A research project that is focused upon the investigation of problems associated with the generation, maintenance, and search of on-line catalogs has been in progress since July 1, 1967 at the Institute of Library Research on the Berkeley Campus of the University of California [78]. This project involves five main aspects. These are the hardware, the software, the file organization, the bibliographic record, and the conversion of existing manual and machine-readable bibliographic records for use in an on-line catalog.

2.4.2 Hardware

The hardware chosen for the project includes an IBM 360/40 with 128K of core memory, an IBM 2314 disk storage unit, an IBM 2740 typewriter terminal, and a Teletype Model 35ASR. These terminals are to be supplemented by IBM 2260 display terminals.

2.4.3 Software

The software is considered to comprise three levels of programs [78]. The first level is a monitor system that

provides for remote terminal operations. The second level encompasses programs that perform the operations of file generation, organization, maintenance, retrieval, and display. The third level consists of programs that monitor and analyze the performance of the system.

On the first level, the Berkeley group reported [78] that it had programmed an experimental terminal monitor system that performs five general functions. These are as follows:

i) Creation of new files.

ii) Retrieval and display of records from existing files.

iii) Compilation of Fortran, PL/I, and Assembler source programs.

v) Interface to user-written programs.

On the second level, batch programs have been written to generate and maintain the master bibliographic file and its associated indexes. This group of programs includes those used to convert existing manual and machine-readable records to the ILR, Institute of Library Research, internal processing format. A data base of 75,000 catalog records has been generated using these programs.

Also on the second level, simple programs for the retrieval and display of bibliographic records have been written. These routines are limited to retrieval by author name, but they can retrieve all records for which the left-most characters of the record key, the author name, are matched by the key supplied in the search request.

On the third level, no programs have been written [78]. However, at this level of programming, much work has been done to implement equivalence-class coding techniques for author names, title words, and other key words. There are two main purposes for the implementation of such equivalence-class coding techniques [78]. These are:

i) Determination of the optimum length of search keys, which affects such things as terminal input-time and index storage requirements.

ii) Provision to the user of a retrieval system with at least some degree of automatic correction of spelling errors.

2.4.4 File Organization

The work done on file organization has been concentrated on the design of a flexible file organization that can be easily modified for experimental purposes [78]. The file design produced by the Berkeley group is basically an inverted file structure with subject, author, and title indexes containing pointers to the appropriate master bibliographic records. Since the bibliographic and index list records are variable in length, while the most efficient packing of mass storage space requires fixed-length physical records of maximum track capacity, techniques have been developed and incorporated into the file design to allow the mapping of variable-length logical records into fixed-length physical records of maximum track capacity.

The index files each consist of two sub-files. The

first sub-file, called the access file, is a keyed file in which the access point is the key and the data field contains a pointer to the second sub-file. The second sub-file, called the address file, is a non-keyed file in which each entry contains a variable number of pointers to the master bibliographic records that have been indexed under the key stored in the access file entry. Logical records are of fixed-length in the access file and of variable-length in the address file.

This partitioning of the index files provides at least two benefits [78]:

i) It is possible to group the records and their principal parts, keys and lists, in several different ways, thus allowing the desired flexibility in the file organization. The more frequently used access file entries can be stored on a low-speed device, such as disk or data cell. This is called horizontal segmentation. Vertical segmentation, the storage of the most-used sets of complete records, as opposed to parts of records, on the faster memory devices, can also be used. Grouping of the access file entries by key length is another possible arrangement. Of course, combinations of these three groupings are also possible.

ii) If the key fields in the access file entries are all of the same fixed-length, then the task of finding a given key is much simplified.

2.4.5 Bibliographic Record

Six different functions are required of bibliographic records in an on-line information retrieval system [78]. These are input, display, printout, processing, communications, and storage. The requirements for these functions are often conflicting. For example, the internal processing function requires ease of manipulation whereas the storage function requires compactness.

The central record designed for the Berkeley project is called the ILR, Institute of Library Research, record and is very similar to the MARC II tape record [111] in both form and content. This facilitates conversion of the MARC II tape records for use in the Berkeley system, and vice versa.

The Berkeley group [78] has given considerable attention to the problems of representing non-standard typographical characters and foreign alphabets and has developed several techniques by which this may be accomplished.

2.4.6 Conversion of Existing Records

A large part of the report by J. C. Cunningham et al. [78] is devoted to a discussion of techniques and costs involved in the conversion of existing manual and machine-readable bibliographic records to the ILR record. Since this area is outside the scope of the present study, it is not discussed further.

2.4.7 Evaluation

The Berkeley group has done a substantial amount of

work towards the solution of the problems involved in the design of bibliographic records for use in an on-line catalog, the representation of non-standard characters and foreign alphabets, the conversion of existing bibliographic data bases, and the implementaiton of equivalence-class coding techniques. As for the problems involved in the design of a file organization for the on-line catalog, they have not, as yet, produced any major breakthroughs; but they have elucidated some fundamental techniques of file organization, including horizontal and vertical segmentation of files, grouping of data elements or records by length, separation of fixed-length and variable-length parts of records, and mapping of variable-length logical records into fixed-length physical records. All of these techniques are applied in Chapters VI and VII of this treatment to the problem of designing a file structure for an on-line integrated library automation system.

2.5 Project INTREX

2.5.1 Introduction

Project INTREX, Information Transfer Experiments, at M.I.T. was conceived at a planning conference in 1965 [87] as a research study aimed primarily at solving the problems of access to documents and document surrogates in an on-line, user-oriented information storage and retrieval system. As the project evolved it became subdivided into two main sub-projects. The first of these was concerned with bibliographic

access to allow the user to search an on-line augmented catalog. The second was concerned with text access to provide the user with the texts of the documents themselves. The augmented catalog sub-project is discussed in more detail below, but the text access sub-project is not discussed here as it is outside the scope of this study.

2.5.2 The Data Base and the Augmented Catalog Concept

The experimental on-line catalog of Project INTREX was planned to encompass bibliographic entries for 10,000 documents in science and engineering [88, 94]. It is augmented in comparison with the standard library card catalog in two ways. First, it includes entries for theses, technical reports, conferences, proceedings, journal articles, and monographs [88]. Thus, it is augmented in its coverage with respect to the ordinary card catalog in which attention is focused on monographs. Second, its entries include many fields of extra information, such as CODEN, abstract, citations, level of approach, and user comments, which are not found in the traditional library catalog [88, 89, 94]. Thus, it is functionally augmented with respect to the conventional card catalog.

A. R. Benenfeld [89] described the augmented catalog's data elements and the techniques used to generate the data base. Benenfeld stated that the augmented catalog entries can accommodate up to 115 data elements combined into 85 fields which in turn are grouped into six major categories, such as descriptive cataloging fields, subject content fields,

catalog control fields, and physical document fields. Data elements within fields are separated by subfield delimiters such as colons, semicolons, parentheses, and brackets. Some data elements are coded, but most are raw text.

2.5.3 Equipment

The version of the INTREX augmented catalog disucssed in this treatment was reported [94] to be operational on an IBM 7094 under M.I.T.'s Compatible Time-Sharing System. Since an experimental, dual-purpose terminal that combines the abilities to display both the disk-stored catalog entries and the microfiche-stored documents pointed to by the catalog entries was not yet available, most of the on-line work reported by R. S. Marcus et al. [94] depended primarily on an IBM 2741. A later report [92] indicated that this special display terminal has since become operational. This report also mentioned that investigations were being carried out to ascertain if it is feasible to implement the augmented catalog system under CP/CMS running on an IBM 360/67.

2.5.4 Subject Indexing

Before going into the details of the file organization and record structures used in the augmented catalog system, it is appropriate to describe the subject indexing of documents as carried out in this phase of the INTREX project.

The INTREX subject indexing philosophy [89, 94] is to use terms based upon the text of the documents being indexed, with each term being a combination of phrases. Each term is

structured to include enough contextual information so that it may stand by itself. Furthermore, each term is weighted to indicate what proportion of the indexed document relates to the represented concept. There is no limit on the number of index terms assigned to a given document, and there is no authority list or thesaurus of approved indexing terms. Instead, a "free vocabulary" based upon the author's own words is used.

2.5.5 File Organization

The files of the augmented catalog system are organized on two levels [88, 89, 94]. The first level consists of the inverted files for author names, title words, and subject terms, together with in-core file directories that serve to localize the position of given sets of records in the disk-stored inverted files. The second level consists of the full catalog records together with a disk-stored catalog directory that is used to translate document numbers, obtained from the inverted files, into the actual disk addresses of the corresponding catalog records. The details of the inverted file, catalog, and catalog directory records are discussed below.

2.5.6 Catalog Records

The catalog records are structured so that the formatting information is separated from the content information [94]. The formatting information, which indicates where a given record or field begins and ends, is stored in a header section. The content information is stored in upper and lower

sections. The upper section contains coded and fixed-length data elements that do not require free formats. The bulk of the information in each catalog entry is stored as raw text in the lower portion of the record, with field and record delimiters and formatting characters removed. Output routines can reinsert these formatting characters to suit the particular line widths of various display devices. The average catalog record structured in this fashion requires from 2000 characters [91] to 2800 characters [94] of storage space. This record structure has a great deal in common with the MARC II [111] and ILR [78] record structures.

2.5.7 Inverted File Records

Each entry in the inverted files is a list of references to catalog records associated with a single primary key, which may be a title or subject word stem, or an author's surname [94]. These references contain not only document numbers but also word specifiers and reference attributes.

The word specifiers indicate which endings were removed to form the word stems. They also indicate the position of the original unstemmed word in the subject phrase and the position of the subject phrase within the subject field. The details of the stemming and phrase decomposition procedures, as well as those of other routines used to generate the inverted files, are found in the reports of R. S. Marcus et al. [94] and A. R. Benenfeld [89].

The attributes include subject-term weights, category, level of approach, and initials of authors' names.

2.5.8 Catalog Directory Records

The catalog directory entries are used to translate document numbers into the physical disk addresses of the corresponding catalog records. As reported in [9], the first ten words of the catalog directory file store general information about the catalog records and the directory itself. The remainder of the directory serves to store pointers to the catalog records with the nth word in this section pointing to the disk record for the nth document.

2.5.9 Searching the Augmented Catalog

The file organization of the augmented catalog permits searches to be conducted on three levels [94]. A first-level search may be made using an author, title, or subject term as the primary key. Matching at this level is enhanced by the previously mentioned word stemming and phrase decomposition procedures that are applied not only to each document's index terms, but also to the subject terms supplied in the user's query. A second-level search can then be carried out using the word specifiers and document attributes as secondary keys. A third-level search is a search through the catalog records themselves.

The search programs permit the user to search the inverted files to locate the catalog entries for all documents referenced by specified author, title, or subject terms , or by combinations of subject, title, and author terms. The search programs provide for the use of "and" logic in queries, but they do not provide for the use of "or" or "not" logic or for

the weighting of terms in queries.

2.5.10 Evaluation

The implementers of the INTREX on-line catalog have concentrated on providing a file organization that can support sophisticated queries that involve complex logic, including "and", "or", "not", "precedence", and "adjacency", and that involve use of stemmed or complete words, surnames or surnames plus initials, weighted subject terms, and level of approach. However, they have not as yet provided a query language that fully exploits these capabilities, nor have they attempted to minimize either the file storage costs or the file access times. For example, they have not attempted to use either manual or automatic classification to group together those catalog entries that are likely to satisfy the same set of queries. Such grouping is likely to reduce the number of disk or data cell accesses that are required to obtain the set of all catalog entries that satisfy any particular query [133].

The use of a free indexing vocabulary, coupled with the absence of a thesaurus that could automatically equate synonyms and near synonyms, does not auger well for the retrieval of all catalog entries that are relevant to any given query, particularly since the absence of "or" logic precludes the inclusion of these synonyms and near synonyms in the query itself. Additionally, the absence of "not" logic hampers the weeding out of false hits.

Despite these shortcomings, the INTREX augmented

catalog project is probably the most advanced of the on-line catalog projects reported in the literature.

2.6 Summary

The foregoing brief review has but touched upon the highlights of a selected group of projects that have involved the design and implementation of on-line library automation systems. All of these projects have resulted in the implementation of one or the other of the three major subsystems, on-line acquisitions, on-line catalog, and real-time circulation, that comprise an on-line integrated library automation system; but none have resulted in the implementation of an on-line system that embodies two or more of these subsystems. Nor does it appear that any of the more ambitious projects mentioned in Section 1.6, among them OCLC [65, 66] and NELINET [67], have produced operational on-line integrated systems.

The next three chapters of this treatment describe an on-line system that encompasses two of the three major subsystems of an on-line integrated system, the on-line catalog subsystem and the real-time circulation subsystem.

CHAPTER III

DESCRIPTION OF IT AND ITS COMMAND LANGUAGE

3.1 Introduction

IT is a single-terminal integrated on-line library automation system which grew out of the work reported by D. M. Heaps and H. S. Heaps [7]. It has been in use in the Reading Room of the Department of Computing Science of the University of Alberta since September 1969.

The current version of IT, in operation since May 15, 1970, includes a real-time circulation subsystem, an on-line patron control subsystem, an on-line catalog search subsystem, an off-line catalog, loans, and author index files maintenance subsystem, and an off-line, as well as an on-line, patron file maintenance subsystem. The automation system, as it now exists, does not include either an acquisition subsystem or a serials subsystem.

The command language for IT is discussed in this chapter; the file organization and record structures are discussed in Chapter IV; and the programs are discussed in Chapter V.

3.2 Data Base

The data base for IT currently consists of the catalog and loans file records for the 1000 books contained in the Reading Room of the Department of Computing Science. It

also includes the patron records for 250 users of the Reading Room. The entries in the catalog file are equivalent to entries in the standard library card catalog. At present, the data base for IT does not encompass records for either bound or unbound journals.

3.3 Equipment

The current version of IT operates under the MTS, Michigan Terminal System, time-sharing system on an IBM 360/67 with 756K of core memory. The computer configuration includes two IBM 2314 disk storage units, two IBM 2301 drum storage units, several tape drives, three card readers, a card punch, three line printers, and telecommunications equipment which handles sixty IBM 2741 typewriter terminals on the campus of the University of Alberta.

IT uses a hard wired IBM 2741 terminal located in the Reading Room of the Department of Computing Science and has 600 4K pages, about 2.5 million bytes, of disk space dedicated to the storage of its files and programs.

A different version of IT was in operation under the CP/CMS time-sharing system from September 1969 until May 15, 1970. Since CP/CMS, unlike MTS, provided only on-line or conversational operation, the version of IT which ran under CP/CMS had only on-line subsystems. This meant that catalog maintenance functions, which are now performed off-line, had to be performed on-line. This was undesirable because these maintenance functions consumed a sizeable portion of the available on-line time.

3.4 Overview of the Command Language

The current version of IT provides nineteen on-line commands which fall into three groups:

i) User-oriented commands.

ii) Circulation commands.

iii) Miscellaneous commands.

The commands in these three groups are listed in Table 3.4.1 and are discussed in Sections 3.6, 3.7, and 3.8. In addition, Appendix I contains an alphabetical index to the commands.

The older CP/CMS version of IT provided an additional eight on-line commands which formed a fourth group, the catalog, loans, and author index maintenance group. The functions of this group are now performed off-line by batch programs.

Before these on-line commands can be discussed in detail, it is necessary to define some basic terminology. This is done in the next section.

3.5 Definition of Terminology

3.5.1 Definition of the Meta Language Used to Describe the Command Language

A meta language is used to describe the commands of the on-line system because it allows the formats of these commands to be defined succinctly and precisely.

The symbols of the meta language are defined as follows:

i) [ ] which enclose required elements of the command language.

TABLE 3.4.1

The On-Line Commands of the IT System

The User-Oriented Commands

SEARCH
RESERVE
BRING
REQUEST
ENQUIRE

The Circulation Commands

LOAN
RETURN
RENEW
FIND
LIST
AVAILABLE
DEMAND
ENROLL
LOOKUP
ALTER
REMOVE

The Miscellaneous Commands

DAY
TRACE
STOP

ii) ' ' which enclose keywords or key symbols of the command language.

iii) ( ) which enclose optional elements of the command language.

iv) = which means that the element on its left is defined by the element or elements on its right.

v) : which means "either, but not both" and is equivalent to the exclusive "or" of boolean logic.

vi) N* which indicates that the element that immediately follows may be repeated from one to N times, where N is a positive integer greater than one.

3.5.2 Basic Definitions for the Command Language

3.5.2.1 Elements Used to Define Author and User Names

[letter] = ['A']:['B']:['C']" ... :['Z']:['a']:['b']" ... :['z']

[digit] = ['0']:['1']:['2']: ... :['9']

[character] = [letter]:[' ']:['-']:['.']

[author surname] = [letter] 18*[character]

[user surname] = [letter] 15*[character]

[initials] = [[[' '][' ']]:[[','](' ')]] 4*[[letter]('.')(' ')]

[corporate name] = [letter] 18*[[character]:[digit]]

3.5.2.2 Definitions of Author Name and Complete Author Name

[personal author name] = [author surname] (initials)

[complete personal author name] = [author surname][initials]

[corporate author name] = [corporate name]

[author name] = [personal author name]:[corporate author name]

[complete author name] = [complete personal author name]:
[corporate author name]

Figures 3.5.1, 3.5.2, and 3.5.3 show examples of author names as defined above.

3.5.2.3 Definition of User Name

[personal user name] = [user surname][initials]

[corporate user name] = [corporate name]

[user name] = [personal user name]:[corporate user name]

Figures 3.5.1 and 3.5.2 show examples of user names as defined above.

3.5.2.4 Definition of User Identification

[user identification] = [user name]:[student identification number]:[social insurance number]:[internal, program-assigned user identification number]

3.5.2.5 Definition of Password

[password] = [number generated by program at start up or by execution of the command DAY]

If the IT system is run on a terminal that has the print suppress option installed, then the password is not printed when it is entered by the librarian.

3.5.2.6 Definition of the Symbol #

The symbol # indicates that the carriage return key on the terminal is to be used.

Smith JPL	von kruger kh
smith, jpl	von kruger, k.h.
SMITH JPL	Stanley-Jones, A.
smith jpl	VON KRUGER KH
smith j.p.l.	
smith, j. p. l.	

Figure 3.5.1 Examples of Valid Personal User Names and Complete Personal Author Names

IBM	bindery
ibm	LOST
Smith-Corona	Interlibrary Loan
smith-corona	C.S. 411
A. B. Dick Corp.	math 420
digital equipment	Dept. of Chemistry

Figure 3.5.2 Examples of Valid Corporate User Names and Corporate Author Names

IBM	von bertlanfy
digital equipment	stanley-jones
Cox, N.S.M.	cox
cox nsm	Von Bertlanfy JP

Figure 3.5.3 Examples of Valid Author Names

3.5.2.7 Definition of the Logical Operators Used with the Command SEARCH

[' '] = ["and" of boolean logic when used between terms]

['|'] - ["inclusive or" of boolean logic]

['¬'] = ["not" of boolean logic]

3.5.2.8 Definition of the Symbol *

['*'] = ["all"]

3.6 The User-Oriented Commands

SEARCH

The command SEARCH is used by both the patron and the librarian to search the catalog file. It may be used to do any of the following:

i) Look up the catalog entry for a particular book, provided that its author and some significant words from its title are known.

ii) Determine the status of a particular book, subject to the conditions specified in i).

iii) Generate a listing of the catalog entries for all books that were written by a particular author or by an author with a particular surname.

iv) Generate a listing of the catalog entries for all books which satisfy the conditions specified in both iii) and iv) above.

The command SEARCH is defined as follows:

[command SEARCH] = ['search'][' '][author][#][title][#]

where:

[author] = [author name]:['*']

[title] = [[(' ')[term]] 3*[[' '](¬')[term]]]:['*']

[term] = [[title word] 3*[['|'][title word]]]

[title word] = 19*[[letter]:[any special character except
for '¬', '|', '*', or ' ']]

These "title words" are not necessarily complete words in the usual sense as they may be right-truncations of complete words. All title words in the catalog file that have the supplied string of characters as their left-most characters are matches as far as the search program is concerned. For example, if the word COMPUTE is specified in the SEARCH command then the program will match on the words COMPUTE, COMPUTER, COMPUTERS, AND COMPUTERIZE, among others.

The search program treats "author names" quite differently. An author name in the catalog file is considered a match if and only if it is identical to the supplied string of characters, except for any punctuation between the surname and the initials or between initials. Such punctuation, if supplied, is ignored by the program.

If the symbol '¬' precedes any term in the title field of the search command, then it applies to every word within that term. For example, "¬DOGS|CATS|FISH" is identical in meaning to "¬(DOGS|CATS|FISH)."

An example of the command SEARCH is:

```
search *
librar computer|automat|system
```

This command asks the search program to find all the catalog entries for books which have titles that contain one of the words LIBRARY, LIBRARIES, LIBRARIAN, and LIBRARIANS, and one of the words COMPUTER, COMPUTERS, COMPUTERIZE, COMPUTERIZED, COMPUTERIZATION, COMPUTERIZING, AUTOMATE, AUTOMATION, AUTOMATIC, AUTOMATING, AUTOMATA, AUTOMATON, SYSTEM, SYSTEMS, SYSTEMATIC, and SYSTEMIC, among others. Obviously, care must be taken in phrasing the title field of the SEARCH command if a deluge of false drops is to be avoided. The "*" in the author field specifies that all authors are acceptable. The results of this command are shown in the sample terminal session in Appendix II.

A second example of the command SEARCH is:

```
search cox
library computer
```

This command asks the program to find all the catalog entries for books that were written by an author whose surname is COX and that have the word LIBRARY and one of the words COMPUTER, COMPUTERS, COMPUTERIZE, COMPUTERIZED, and COMPUTERIZATION.

A third example of SEARCH is:

```
search cox n s m
*
```

This command asks the program to find the catalog entries for all books that were written by an author whose surname is COX

and whose initials are NSM.

A fourth and final example is:

search *

statistic theory ¬information|decision|gambling|communication

This command asks the search program to find the catalog entries for all titles which contain the word THEORY and one of the words STATISTIC, STATISTICS, and STATISTICAL, but which do not contain any of the words INFORMATION, DECISION, GAMBLING, and COMMUNICATION, or any of the words for which these four words are right-truncations.

RESERVE

If the command SEARCH has found a unique match for a patron's query, then the patron may reserve the book by use of the command RESERVE. If the book is available in the library, or if the reservations queue for the book is full, then the reservation is refused. If the book is not available and the reservations queue is not full, then the program asks the patron to type in his identification, which may be any one of the four types defined in Section 3.5.2.4. A patron is not allowed to reserve a book if he already has it on loan and is not allowed to place more than one reservation against a book. However, special classes of "patrons", such as BINDERY, INTERLIBRARY LOAN, LOST, and CS 411, may have multiple copies of a book on loan and may place multiple reservations against books, including those that are already on loan to them. After entering the reservation into the system's files, the program types a message which tells the

patron where he stands in the waiting list for the book.

The command RESERVE is defined as follows:

[command RESERVE] = ['reserve'][#]

BRING

Similar to the command RESERVE, the command BRING is used only after the command SEARCH has found a unique match for the patron's query. It instructs the library to fetch the book for the patron and is intended for use in an environment in which terminals are in several branch libraries as well as in the main library.

BRING is defined as follows:

[command BRING] = ['bring'][#]

REQUEST

The command REQUEST may be used instead of RESERVE to place a reservation against a book. It is normally more convenient than RESERVE since it does not require prior use of the command SEARCH. Otherwise, it is subject to the same restrictions as RESERVE.

REQUEST is defined as follows:

[command REQUEST] = ['request'][' '] [complete author name][#]
[call number][#][user identification][#]

ENQUIRE

A patron may use the command ENQUIRE to determine if a previously reserved book is available. ENQUIRE has the same format as REQUEST with 'request' replaced by 'enquire'.

3.7 The Circulation Commands

The commands in this group are used by the librarian to perform functions related to the circulation of books, including the maintenance of patron records. Patrons are not allowed to use these commands, which are protected by a password that is normally known only to the librarian.

LOAN

The command LOAN is used to enter a loan transaction into the system's files. Patrons, other than the special ones, such as LOST, BINDERY, and CS 411, may not borrow more than one copy of a book.

LOAN is defined as follows:

```
[command LOAN] = ['loan'][' '][complete author name][#]
                 [call number][#][user identification][#][password][#]
```

RETURN

The command RETURN is used to enter a return transaction into the system's files. If there are any reservations placed against the book then the program types a message which asks the library to hold the book for the patron who is first on the waiting list for the book. If the returned book is on the patron's chain of overdue books in file OVERDUES then it is removed from that chain. RETURN has the same format as LOAN with 'loan' replaced by 'return'.

RENEW

The command RENEW is used to renew the loan of a book. The program does not allow a renewal if there are any

reservations placed against the book. If the renewed book is on the patron's chain of overdue books in file OVERDUES then it is removed from that chain. RENEW has the same format as LOAN with 'loan' replaced by 'renew'.

FIND

The command FIND provides the librarian with the following information about a particular book:

i) The number of copies held by the library.

ii) The number of copies currently on loan to patrons.

iii) The name and internal identification number of the patron who has each copy that is on loan and the date that the copy was borrowed.

iv) The names and internal identification numbers of all patrons who have placed reservations against the book. Also their positions in the waiting list for the book.

The command FIND also provides the librarian with the option of deleting one or more reservations from the waiting list for the book.

FIND is defined as follows:

[command FIND] = ['find'][' '][complete author name][call number][#][password][#]

LIST

The command LIST is used to generate a listing of all the books that are currently on loan to a particular user. This listing gives the author and L.C. call number of each

book and indicates whether or not it is overdue or reserved.

LIST is defined as follows:

[command LIST] = ['list'][#][user identification][#][password][#]

AVAILABLE

The command AVAILABLE is used to generate a listing of all books that have reservations placed on them and that are currently available to the first person on their waiting list. Each entry in the listing of available books gives the author and L.C. call number of the book and the name of the user who is first on the waiting list for it.

AVAILABLE is defined as follows:

[command AVAILABLE] = ['available'][#][password][#]

DEMAND

The command DEMAND is used to search the file LOANS for overdue books. If any books are found to be two weeks or more overdue, then the program generates overdue notices and stores them in the file OFFLINE.I/O for later printing off-line on the high-speed line printer or on-line on the terminal. This is discussed further in Section 4.7. The execution of DEMAND also causes records for overdue books to be added to the file OVERDUES of which more is said in Section 4.3.3.

During the execution of DEMAND, if a reserved book is found to have been available to the first patron on the book's waiting list for more than three days, then the program will delete the reservation for that patron and move

up the remaining reservations.

After the program completes its scan of the loans file, it prints a summary on the terminal. This summary gives the number of books that are overdue four or more weeks, the number of overdue notices generated, and the number of reservations deleted. If any books were found to be four or more weeks overdue, then the program lists, on the terminal, the author, the L.C. call number, the patron identification number, and the number of weeks overdue for each of them.

DEMAND is defined as follows:

[command DEMAND] = ['demand'][#][password][#]

ENROLL

The command ENROLL is used to enroll new library patrons. After the command has been entered from the terminal, the program types out a message which asks the librarian to type in the surname, initials, title code, privilege code, address, phone number, departmental code, and student identification number or social insurance number for each new patron. These data elements are discussed further in Section 4.3.1 and Section 4.6.

After the data for each new user has been entered, it is checked by the program for errors. If the data is correct, then it is written into the file USERS and it is echoed back to the terminal so that the librarian can visually check the record that has been entered in to the file USERS. The program also types out the internal user identification number that it has assigned to the new patron. The program is then

ready to accept the data for the next new patron.

If the program detects any errors in the data entered for a new patron, then it types out appropriate error messages, echoes the entry back to the terminal, and asks the librarian to reenter the data.

If the required information has been entered for all the new patrons, then the librarian enters an asterisk, "*", to indicate this to the program. The program then completes processing of the new entries and updates the file USERINDEX, which is discussed in Section 4.3.2. The program also checks for duplicate names, duplicate student identification numbers, and duplicate social insurance numbers. If any such duplicates are found, then appropriate error processing is performed.

Figure 4.6.2 summarizes the actions of the command ENROLL.

The command ENROLL is defined as follows:

[command ENROLL] = ['enroll'][#][password][#]

LOOKUP

The command LOOKUP is used to display the record of a particular patron. It is defined as follows:

[command LOOKUP] = ['lookup'][#][user identification][#]
[password][#]

ALTER

The command ALTER is used to change certain parts of a patron's record in the file USERS. The data elements that

may be altered are the surname, the initials, the title code, the privilege code, the address, the departmental code, and the student identification number or social insurance number. The title code and the privilege code are defined in Section 4.6.1. After the command ALTER has been entered at the terminal, the program displays these data elements of the patron's record and then asks the librarian to enter the desired changes. ALTER has the same format as LOOKUP with 'lookup' replaced by 'alter'.

REMOVE

The command REMOVE is used to remove a patron's record from the file USERS. After the command has been typed in at the terminal, the program displays the patron's record and then asks the librarian to confirm that it is indeed the correct record to be removed. If the librarian confirms that the displayed record is the one that she wishes removed, then the program proceeds to purge the record. REMOVE has the same format as LOOKUP with 'lookup' replaced by 'remove'.

3.8 The Miscellaneous Commands

DAY

The command DAY is used to override the current date provided to the on-line subsystem LIBRARY by the MTS command TIME, which is automatically called whenever LIBRARY is started. At any time while LIBRARY is in operation, the librarian can enter the command DAY and alter the current

date to some past or future date. After the command has been entered, the program asks the librarian to type in the number of the new day, with January 1, 1968 taken as day number one. The program then calculates a new password which replaces the password previously calculated when LIBRARY was started. DAY is most useful when the programs are being tested.

DAY is defined as follows:

[command DAY] = ['day'][#][password][#]

TRACE

The command TRACE is intended primarily for use by the programmer as a debugging aid. TRACE is entered just prior to the entry of the command which is to be traced.

TRACE is defined as follows:

[command TRACE] = ['trace'][#][password][#]

STOP

The command STOP is used to stop the execution of the on-line program. It is defined as follows:

[command STOP] = ['stop'][#][password][#]

3.9 Prompting

Since human beings are prone to forgetfulness, it was thought wise to install extensive prompting freatures in the programs that handle the on-line commands discussed above. If the librarian or the patron is entering a command and is unsure of what is required next, then he can type in a question mark, "?", and the program will type out a message which indicates what data should be entered next.

Figure 3.9.1 illustrates the entry of a command without prompting and Figure 3.9.2 illustrates the entry of the same command with prompting. The program always types out the message "ENTER A COMMAND" when it is ready to accept a command from the terminal. In addition, messages typed out by the program are always in upper case.

```
ENTER A COMMAND
loan flores i
qa76.5f63
(password entered, but not printed on the terminal)
TRANSACTION COMPLETED
ENTER A COMMAND
```

Figure 3.9.1 Entry of a Command Without Prompting

```
ENTER A COMMAND
loan flores i
?
ENTER CATALOG NUMBER
qa76.5f63
?
ENTER USER IDENTIFICATION
jones jp
?
ENTER PASSWORD
(Password entered, but not printed on the terminal)
TRANSACTION COMPLETED
ENTER A COMMAND
```

Figure 3.9.2 Entry of a Command With Prompting

CHAPTER IV

THE FILE ORGANIZATION AND THE RECORD STRUCTURES OF IT SYSTEM

4.1 Introduction

The current MTS version of the IT system operates on the fifteen logical disk files listed in Figure 4.1.1. Ten of these files are used by the on-line subsystems. Of these ten files, seven are primary files and three are secondary or auxiliary files. The primary files are shown in Figure 4.1.2. The remaining five of the fifteen files are auxiliary files used exclusively by the batch-mode programs that maintain the seven primary files. Appendix III contains an alphabetical index to these logical disk files.

The fifteen logical disk files may be grouped, somewhat arbitrarily, into six functional groups as follows:

i) The Catalog and Loans Group. This group consists of the three files named AUTHORS, CATALOG, and LOANS, and forms the data base for the on-line catalog search and on-line book control subsystems.

ii) The Patron Group. This group consists of the files named USERS, USER NAME INDEX, USER ID NUMBER INDEX, and OVERDUES, and forms the data base for the patron control subsystem. The files USER NAME INDEX and USER ID NUMBER INDEX are both subfiles of the MTS line file called USERINDEX.

AUTHORS
CATALOG
DELETIONS
FREELOCS
FREESTOR
LOANS
NEWBOOKS
NEWENTRIES
NEWUSERS
OFFLINE.I/O
OVERDUES
USER ID NUMBER INDEX
USER NAME INDEX
USERS
SAVEFILE

Figure 4.1.1 The Logical Disk Files Operated on by the IT System

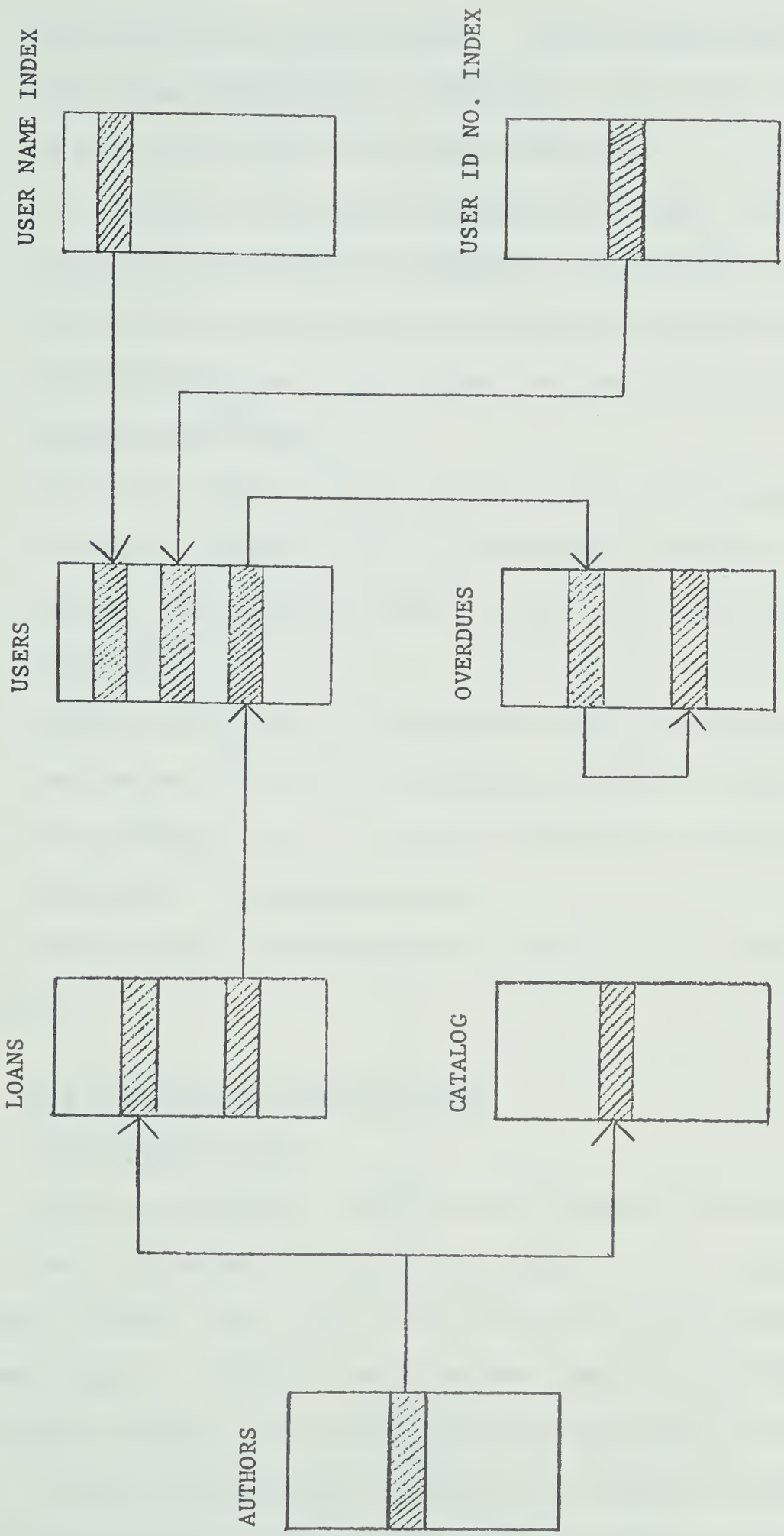


Figure 4.1.2 The Primary Files of the On-Line Subsystems

iii) The Patron Auxiliary Group. This group consists of the files FREESTOR and FREELOGS, which are subfiles of the MTS line file named FREESPACE.

iv The Catalog and Loans Maintenance Group. This group consists of the files NEWBOOKS, NEWENTRIES, DELETIONS, and SAVEFILE, which are used by the program module BATCHUPDATE when it is updating the files AUTHORS, CATALOG,and LOANS.

v) The Patron Maintenance Group. This group consists of just one file, the file NEWUSERS, which is used during the batchmode updating of the files USERS and USERINDEX.

vi) The Utility Group. This group also consists of just one member, the file OFFLINE.I/O, which is used mainly as a system log. It is also used for the temporary storage of overdue notices.

Each of these groups is discussed in turn in the sections that follow.

4.2 The Catalog and Loans Group

4.2.1 The Catalog File

The file CATALOG, which is the central bibliographic file of the IT system, is shown in Figure 4.2.1, along with the other two MTS line files that are part of the Catalog and Loans Group. CATALOG contains one complete extended bibliographic record for each author of each book in the system. These bibliographic records are termed "extended" because they contain all the information found in the

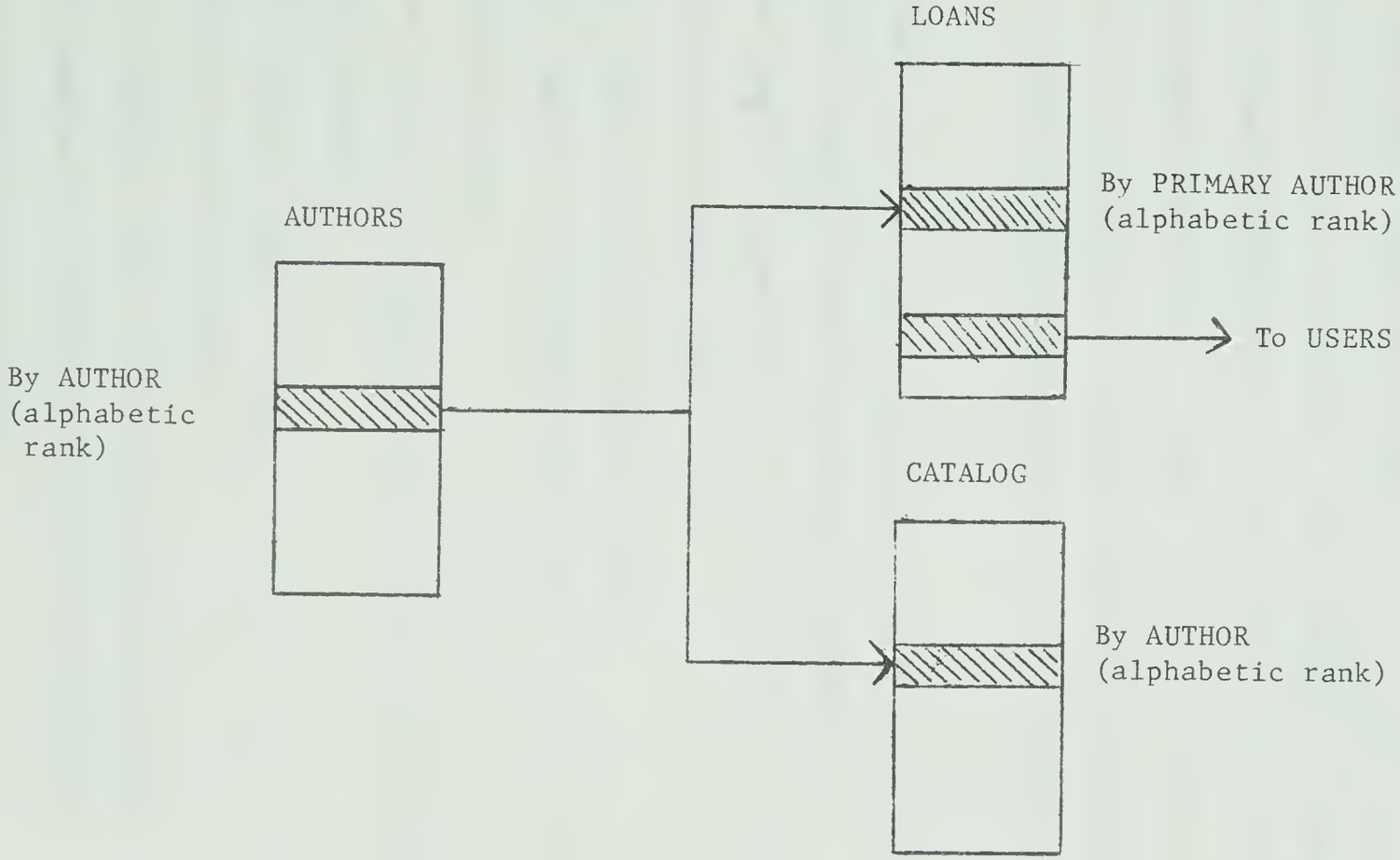


Figure 4.2.1 The Files AUTHORS, LOANS, and CATALOG

standard library card catalog entries plus additional information not normally found in a card catalog. This additional information includes purchase order number, fund number, cost, and date received. It has been included to allow the catalog file to be used as the basic file for a future acquisitions subsystem. All entries in CATALOG contain, in their first 24 bytes, a 24-byte key field which in turn contains an author surname and up to four initials. The author surname in this key field is right truncated, if necessary to 19 characters. This author-name key field is used (i) to sort the entries in CATALOG into alphabetical order according to author name and (ii) as the primary access point to the entries in CATALOG.

In addition to the information mentioned above, each catalog entry contains the following information:

i) The complete names of all authors of the book.

ii) The title of the book.

iii) The subtitle of the book, if any.

iv) The imprint: the publisher and the place and date of publication.

v) The collation: pagination, illustration statement, size, price.

vi) The series note, if any.

vii) The edition statement.

viii) The library that currently holds the book.

ix) The number of copies held by the library.

x) The loan period in days.

xi) The accession numbers of up to four copies of the book.

It should be noted at this point that the term authors is to be interpreted as meaning not only authors as they are usually defined by the librarian, but also editors, compilers, revisers, translators, and committee chairmen, among others.

Each catalog entry consists of three or more 80-byte logical records, each of which occupies one line in the MTS line file CATALOG. These catalog entries are structured as follows:

i) The first 80-byte record:

Bytes	Field No.	Content of the Field
1-20	1	Author surname, left-justified and right-truncated to 19 characters, if necessary.
21-24	2	Up to four initials, left-justified and with no punctuation. Fields 1 and 2 comprise the author-name key field referred to above.
25-26	3	Two digits that indicate the number of letters in the author surname contained in Field No. 1.
27-28	4	Two digits that indicate the number of searchable title words contained in Field No. 19.
29-46	5	Nine two-digit numbers that indicate the positions of the first letters of the first nine title words relative to the beginning of the second 80-byte record in the catalog entry.

Bytes	Field No.	Content of the Field
47-78	6	The author details for the author whose name is used in Fields 1 and 2. These author details include surname, forenames or initials, dates, epithets, and relator.
79-80	7	Two digits that indicate the number of 80-byte middle records that follow. These middle records contain the title, the complete author statement, the imprint, and certain other items.

The line labelled 1149 in Figure 4.2.1.1, shows a sample first record. The numbers above the line refer to the numbered fields described above.

ii) The middle 80-byte records:

These records, of number given in Field 7, contain up to nine searchable words of title, separated by single blanks, followed by two consecutive blanks which are followed by the remainder of the title, if any. The first letter of the first searchable title word is placed in the first byte of the first of these records. The title is followed in order by the subtitle, if any; the complete author statement, if the book has more than one author; the series statement, if any; the imprint; the collation; and any other pertinent information that is not included elsewhere. All information that follows the searchable title words is in free format. These middle records are illustrate by the lines labelled 1150 through 1154 in

```
1149  COX                      NSM | 3| 5| 1| 5|14|18|22| 0| 0| 0| 0|COX, NIGEL S. M.                    | 5|
1150  |THE COMPUTER AND THE LIBRARY,| THE ROLE OF THE COMPUTER IN THE   ORGANIZATION A
1151  ND HANDLING OF INFORMATION IN LIBRARIES,| BY        N. S. M. COX, J. D. DEWS, AN
1152  D J. L. DOLBY.|                          (UNIVERSITY LIBRARY PUBLICATIONS, NEWCASTL
1153  E UPON TYNE, 4)          UNIVERSITY OF NEWCASTLE UPON TYNE LIBRARY, 1966, 95 P,|
1154  | 24 CM C.4                                                                        |
1155  |TZ678.9C87        | 1| 1|14| 2|697499|     0|     0|     0| 0 0 0|   0| 0|  0.0|
```

```
1327  DEWS                     JD   4 5 1 5141822 0 0 0 0DEWS, J. D.                    5
1328  THE COMPUTER AND THE LIBRARY,  THE ROLE OF THE COMPUTER IN THE    ORGANIZATION A
1329  ND HANDLING OF INFORMATION IN LIBRARIES,          BY N. S. M. COX, J. D. DEWS,
1330   AND J. L. DOLBY.                        (UNIVERSITY LIBRARY PUBLICATIONS, NEWCASTL
1331  E UPON TYNE, 4)          UNIVERSITY OF NEWCASTLE UPON TYNE, 1966, 95 P., 24 CM.
1332  , C.4.
1333  TZ678.9C87        1 0 0 2697499     0     0     0 0 0 0     0 0  0.0
```

Figure 4.2.1.1 Sample Entries from the File CATALOG

Figure 4.2.1.1. The searchable title words are contained in the field labelled 19 in line 1150.

iii) The last 80-byte record:

Bytes	Field No.	Content of the Field
1	8	The end flag, which is "F" if the entry is the last one in the file CATALOG, otherwise "T."
2-17	9	Sixteen characters that give the Library of Congress call number for the book. The call number is left-justified within this field and is right-truncated, if necessary, to 16 characters.
18-19	10	A two-digit code that denotes the library that holds the book. The only code assigned so far, "01," denotes the Computing Science Reading Room.
20-21	11	Two digits which indicate the number of copies of the book that are held by the library. The number of copies is set equal to zero if the catalog entry is for a secondary author. Secondary authors, as well as primary authors, are defined in Section 4.2.2.
22-23	12	Two digits which indicate the loan period in days. This field contains zero if the entry is for a secondary author or for a reference work.

Bytes	Field No.	Content of the Field
24-25	13	Two digits which indicate the number of authors of the book in addition to the one given in Fields 1, 2, and 6.
26-53	14	Four seven-digit fields that contain the accession numbers of up to four copies of the book. If these accession numbers are less than seven digits in length, then they are right-justified within their individual seven-digit fields.
54-59	15	The date the book was received by the library, in the form YYMMDD where the YY are two digits that indicate the year, the MM are two digits that indicate the month, and the DD are two digits that indicate the day of the month.
60-65	16	Six digits that indicate the number of the purchase order for the book.
66-65	17	Two digits that indicate the number of the fund that paid for the book.
68-70	18	The cost of the book in the form ddd.cc where the ddd indicate the number of dollars in the cost and the cc indicate the number of cents in the cost.
71-80		Unused at present.

A sample catalog entry is shown in Figure 4.2.1.1, the numbers above or below each line correspond to the fields

discussed above. The number to the left of each line is that line's line number in the MTS line file CATALOG. It is important to note that lines 1154 and 1155 shown in Figure 4.2.1.1 do not occupy a full 80 bytes in the MTS line file CATALOG because the MTS file handling routines delete all except one of the trailing blanks after the last non-blank character in each line before storing the line in the disk file [134]. When a line is read from a disk file, these file handling routines place it, left-justified, into a buffer of appropriate length, in the case of CATALOG 80 bytes, and then fill the remainder of the buffer with blanks.

4.2.2 The Loans File

This file contains one 77-byte loan and reservations record for each copy of each book in the system. Each record consists of one line in the MTS line file LOANS. The records in LOANS are arranged in alphabetical order according to primary author name. When a book has more than one author, the primary author is defined to be the author whose name is listed first on the title page of the book. There are no entries in LOANS for the other authors of the book, hereafter known as secondary authors.

The entries in LOANS are structured as follows:

Bytes	Field No.	Content of the Field
1-20	1	Primary author's surname, left-justified and right-truncated, if necessary, to 19 characters.
21-24	2	Primary author's initials, left-justified

Bytes	Field No.	Content of the Field
		and with no punctuation.
25-26	3	A two-digit number which indicates the length of the author surname contained in Field No. 1.
27-42	4	LC call number, left-justified and right-truncated, if necessary, to 16 characters.
43-48	5	A six-digit field that contains the internal user identification number of the user who currently has this copy of the book on loan. If this number is less than six digits in length, then it is right-justified within the field. The internal user identification number, which is assigned by the subroutine ENROLL when the user is enrolled, is the line number of the user's record in the file USERS and, therefore, is a direct pointer to that record.
49-66	6	Three six-digit fields that contain the internal user identification numbers for up to three users who have requested the book. If less than six digits in length, these numbers are right-justified in their individual fields.
67-70	7	A four-digit number that indicates, if non-zero, the date on which this copy of the book was borrowed, if it is currently

Bytes	Field No.	Content of the Field
		on loan, or the date on which it was returned, if it is not currently on loan. If this date number is zero, then this copy of the book has never been borrowed. This date number indicates the day relative to January 1, 1968, which is taken as day number one.
71-73	8	A three-digit number which indicates the total number of times that this copy of the book has been borrowed since it was entered into the system's files.
74-75	9	Two digits that indicate the number of copies of the book.
76-77	10	Two digits which indicate the loan period of the book in days.

A sample entry from LOANS is shown in Figure 4.2.2.1. The numbers above the line correspond to the numbered fields discussed above.

4.2.3 The Author Index File

This file, named AUTHORS, is an alphabetically arranged index to the entries in the files LOANS and CATALOG. It contains one 35-byte entry for each author of each book in the system. Each entry consists of the author's name and pointers to the first records of the first relevant entries in the files LOANS and CATALOG. Each index entry is one line in the MTS file AUTHORS.

	1	2	3	4	5	6			7	8	9	10
203	COX	NSM	3Z678.9C87		232	0	0	0	1163	4	114	

Figure 4.2.2.1 Sample Entry from the File LOANS

	1	2	3	4
172	COX	NSM	203	1149

Figure 4.2.3.1 Sample Entry from the File AUTHORS

These index entries in AUTHORS have the following structure:

Bytes	Field No.	Content of the Field
1-20	1	The author's surname, left-justified and right-truncated, if necessary, to 19 characters.
21-24	2	The author's initials, left-justified and with no punctuation.
25-29	3	A five-digit pointer to the first record in the loans file, LOANS, that contains the author's name. This pointer is the line number of the indicated record in LOANS. If the author appears only as a secondary author, then this pointer is zero.
30-35	4	A six-digit pointer to the first 80-byte record, or line, of the first catalog entry in file CATALOG that has this author name in its author-name key field.

a sample index entry from AUTHORS is shown in Figure 4.2.3.1. The numbers above the line correspond to the numbered fields discussed above.

4.3 The Patron Group

The four files that comprise the Patron Group, together with the two auxiliary files FREELOCS and FREESTOR, are shown in Figure 4.3.1. The two index files, USER NAME INDEX and USER ID NUMBER INDEX, are also called, respectively, NAMEKEY

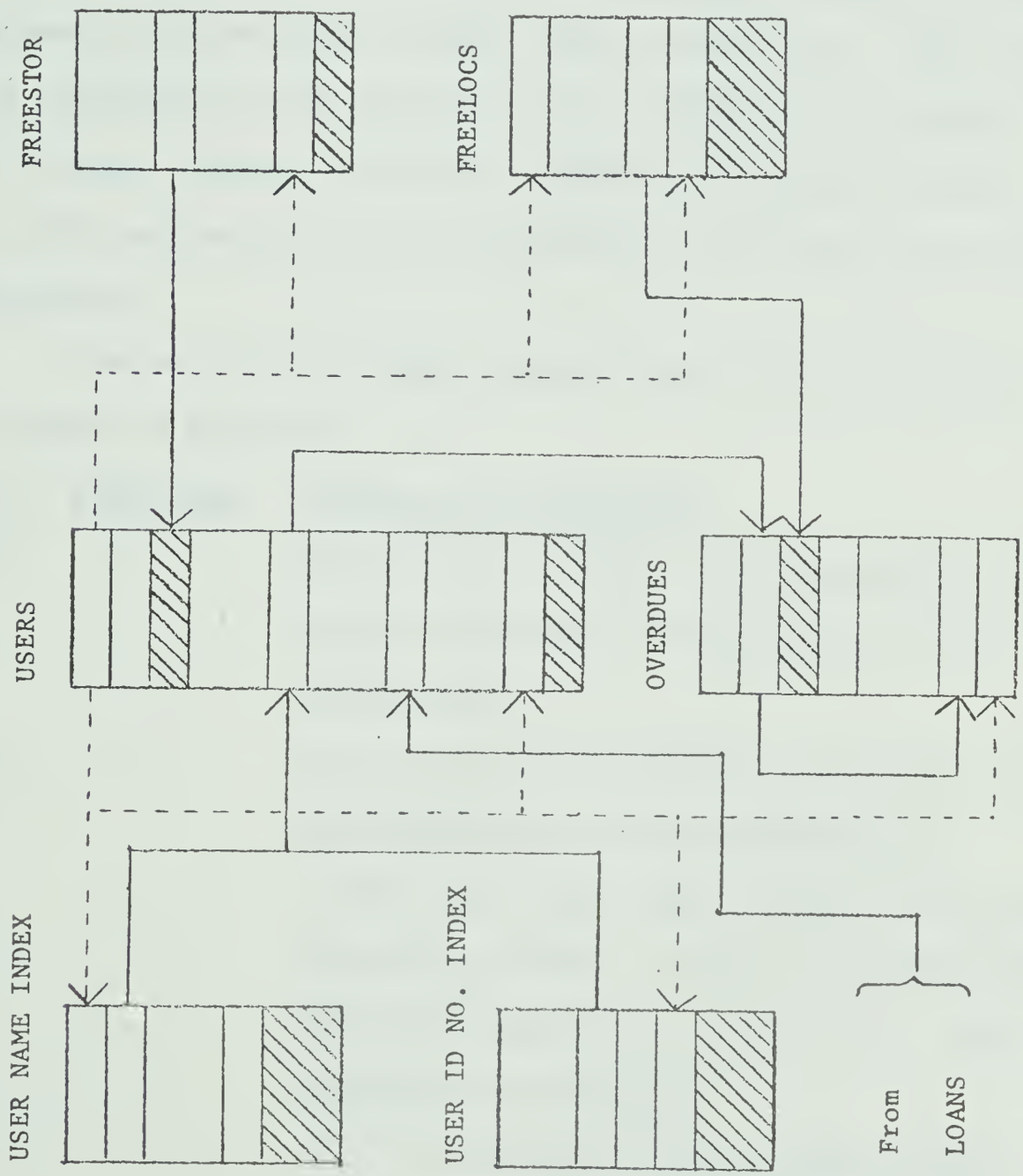


Figure 4.3.1 The File USERS and Related Files

and IDNKEY after they have been read into core and set up as in-core index tables.

4.3.1 The File USERS

The MTS line file USERS contains one 80-byte record for each user who is currently enrolled in the system. The records in USERS are unordered; a new user's record is simply inserted into the first vacant slot in the file. Each user's entry consists of one 80-byte line in USERS and contains the user's name, address, identity numbers, telephone number, title and borrowing status indicators, and other pertinent information.

The entries in USERS, apart from the first record, are structured as follows:

Bytes	Field No.	Content of the Field
1-16	1	The user's surname, left-justified and right-truncated, if necessary to 16 characters.
17-20	2	Up to four of the user's initials, left-justified and with no punctuation.
21	3	A one-digit code that denotes the user's borrowing status. This is the privilege code referred to in Section 3.7. The codes are as follows: i) "1" indicates faculty privileges. ii) "3" indicates non-academic staff privileges.

Bytes	Field No.	Content of the Field
		iii) "4" indicates graduate student privileges. iv) "7" indicates undergraduate student privileges. v) "8" indicates special privileges for "users" such as MISSING, BINDERY, INTERLIBRARY LOAN, and courses. iv) The remaining five codes are unassigned.
22	4	A one-digit code that denotes the title of the user. The assigned codes are as follows: i) "1" indicates Mr. ii) "2" indicates Mrs. iii) "3" indicates Miss. iv) "4" indicates Dr. v) "5" indicates Prof.
23-46	5	A 24-character field that contains the user's address in free format.
47-53	6	A seven-character field that contains the telephone number.
54-55	7	A two-letter code which denotes the user's department or faculty. These codes are the standard ones used by the university's administration. A sample code is "CS" which denotes the Department of Computing Science.

Bytes	Field No.	Content of the Field
56-64	8	A nine-digit field that contains either the user's nine-digit Social Insurance Number or the user's six-digit Student Identification Number, which is right-justified.
65	9	A one-digit code that indicates the status of the user's account with the library. The assigned codes are as follows: i) "0" indicates that the user is active, but has no books on loan. ii) "1" indicates that the user has books on loan and that none of them are overdue. iii) "2" indicates that the user has books on loan and that some of them are overdue. iv) "3" indicates that the user is blacklisted. v) "4" indicates that the user is inactive. When fully enabled, the patron control routines automatically prevent users from signing out books if their account status is "3" or "4."
66-69	10	A four-digit number which indicates the date of the user's enrollment in the system. This date number indicates the

Bytes	Field No.	Content of the Field
		date relative to January 1, 1968, which is taken as day number one.
70-71	11	Two digits that indicate the number of books currently on loan to the user.
72-73	12	Two digits that indicate the number of books that the user has overdue.
74-76	13	A three-digit pointer to the beginning of the user's list of overdue books, if the user has any books overdue. If the user has no books overdue then the pointer is zero. This pointer is the line number of the first record of the user's overdue list in the MTS line file OVERDUES.
77-80	14	Four digits that indicate the line number of this record in the MTS line file USERS. This line number is the system's internal identification number for this user and it is this internal identification number that is used in Fields 5 and 6 of the file LOANS, which is discussed in Section 4.2.2. If a transaction entered from the terminal supplies the user's name, Social Insurance Number, or Student Identification Number, then the supplied user identification must be translated into the user's internal identification number

Bytes	Field No.	Content of the Field
		before the transaction can be entered into the file LOANS. This translation is performed by table look-up on the in-core user name and ID number index tables, which are discussed in Section 4.3.2.

A sample entry from the file USERS is shown in Figure 4.3.1.1. The numbers above the line correspond to the numbered fields discussed above.

The first record, or line, in the file USERS does not contain a user's record as described above, but instead contains certain system parameters. This 20-byte record is structured as follows:

Bytes	Field No.	Content of the Field
1-4	1	Four digits that indicate the number of occupied records in the MTS line file USERINDEX, which is described in Section 4.3.2. This number is also the number of active users enrolled in the system because USERINDEX contains one entry for each active user.
5-8	2	A four-digit pointer to the last record in the file USERS.
9-12	3	A four-digit pointer to the last non-empty record in the file FREESTOR, which is described in Section 4.4.1.
13-16	4	A four-digit pointer to the last non-empty record in the file FREELOCS, which is

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
68	DIMSDALE	JJF	4	1	305-10621-79AVE.	3979	CS	642733	0	626	0	0	0	68

Figure 4.3.1.1 Sample Entry from the File USERS

	1	2	3	4	5
54	DIMSDALE	JJF	68	641932	147

	1	2	3	4	5
46	CS580		7	622978	213

Figure 4.3.2.1 Sample Entries from the File USERINDEX

Bytes	Field No.	Content of the Field
		described in Section 4.4.2.
17-20	5	A four-digit pointer to the last record in the file OVERDUES, which is described in Section 4.3.3.
21-24	6	A four-digit pointer to the first record in the file FREELOCS.

4.3.2 The User Index Files and Tables

The MTS line file USERINDEX contains a pair of subfiles which are indexes to the file USERS, which is described in Section 4.3.1. One subfile, USER NAME INDEX, is sequenced alphabetically by user name; the other, USER ID NUMBER INDEX, is sequenced numerically by user identification number, which may be either a social insurance number or a student identification number. The file USERINDEX is read into core memory by the subroutine SETUP when the on-line program module LIBRARY is started and becomes the two in-core index tables. The first index table, NAMKEY, is arranged in alphabetic order according to user name. The second index table, IDNKEY, is arranged in numeric order according to user identification number.

If the program needs to refer to a user's record and it has been supplied with the user's name, social insurance number, or student identification number, then it first performs a binary search on one of these in-core index tables to obtain a pointer to the required patron record, which is then read into core. Since the pointer thus obtained is not only

the line number of the user's record in the file USERS, but is also the user's internal identification number, the translation required for transactions that involve the file LOANS is also effected by this table look-up procedure.

If the program has been supplied with the internal identification number of the user, then it bypasses the binary search on these index tables and reads the required record directly into core.

Each 40-byte entry in the file USERINDEX contains one 25-byte entry from the subfile USER NAME INDEX and one 15-byte entry from the subfile USER ID NUMBER INDEX. These two entries are not ordinarily for the same user because the two subfiles are sorted independently. Each entry in USERINDEX is structured as follows:

Bytes	Field No.	Content of the Field
1-16	1	The user's surname, left-justified and right-truncated, if necessary, to 16 characters.
17-20	2	The user's initials, up to four, left-justified and with no punctuation.
21-25	3	A five-digit pointer to the user's record in the file USERS.
26-35	4	Up to ten digits, right-justified within this ten-byte field, which are the user's social insurance number or student identification number.
36-40	5	A five-digit pointer to the user's record in the file USERS.

Fields 1, 2, and 3 comprise the USER NAME INDEX entry and Fields 4 and 5 comprise the USER ID NUMBER INDEX entry. Figure 4.3.2.1 shows two sample entries from the file USER-INDEX; the numbers above the lines correspond to the fields discussed above.

4.3.3 The File OVERDUES

The MTS line file OVERDUES contains chained lists of overdue books. If a user has any books overdue, then his record in the file USERS contains a pointer to an overdue book record in the file OVERDUES, and this record itself has a pointer to the next record in this user's list of overdue books.

The file OVERDUES is maintained by the subroutine DEMAND, which adds records to the chained lists, and by the subroutine RESERV, which deletes records from the chained lists. Both RESERV and DEMAND accomplish these tasks by calling the subroutine LBOOK which does the actual list manipulation.

Each 25-byte record in OVERDUES is structured as follows:

Bytes	Field No.	Content of the Field
1-16	1	This 16-byte field contains the LC call number of the overdue book. The call number is left-justified and right-truncated, if necessary to 16 characters.
17-20	2	This four-byte field contains a right-justified number that indicates the date on which the book was borrowed. The form of this date is described in Section 4.2.2.

Bytes	Field No.	Content of the Field
		in the discussion of Field No. 7 of the records in the file LOANS.
21-23	3	This three-byte field contains a right-justified number of up to three digits, which is a pointer to the next record in the user's list of overdue books. The pointer is zero if this record is the last one in the list.
24-25	4	This two-byte field contains a two-digit number which indicates the number of weeks that the book has been overdue.

4.4 The Patron Auxiliary Group

The two files that comprise the Patron Auxiliary Group are shown in Figure 4.3.1. Both of these files, FREESTOR and FREELOCS, are subfiles of the MTS line file FREESPACE, with FREESTOR occupying the upper portion of FREESPACE and FREELOCS occupying the lower portion. The The boundary between these two subfiles is defined by Field No. 6 of the first record in the file USERS, which is described in Section 4.3.1. If necessary, the subroutine SETUP automatically shifts FREELOCS downward within FREESPACE so as to allow FREESTOR to expand.

4.4.1 The File FREESTOR

The file FREESTOR contains a list of vacant slots in the file USERS. It allows the subroutine ENROLL to insert

the record for a new user into the space formerly occupied by the record for a user who is now inactive. Whenever a user is removed from the system's files by the command REMOVE, a pointer to the user's record is added to the list of such pointers in the file FREESTOR. New user's records are always placed into any vacant slots before the file USERS is extended by the addition of new records to its lower end. The last non empty record in FREESTOR is read into core memory by the subroutine SETUP at the start-up of the on-line program module LIBRARY. This last record thus becomes an in-core list of vacant slots in the file USERS.

Each 80-byte record in FREESTOR is structured as follows:

Bytes	Field No.	Content of the Field
1-4	1	Four digits which indicate the number of non-zero pointers, up to 19, that are contained in the rest of the record.
5-80	2	A list of up to 19 four-digit pointers to vacant slots in the file USERS.

4.4.2 The File FREELOCS

This file contains a list of vacant slots in the file OVERDUES and serves the same function in relation to OVERDUES that FREESTOR serves in relation to USERS. The last non-empty record in this file is read into core memory by subroutine SETUP at system start-up time, and thus becomes an in-core list of vacant slots in OVERDUES. FREELOCS is maintained by subroutine LBOOK.

Each 80-byte record in FREELOCS is structured as follows:

Bytes	Field No.	Content of the Field
1-2	1	Two digits which indicate the number of non-zero pointers, up to 26, that are contained in the remainder of the record.
3-80	2	A list of up to 26 three-digit pointers to vacant slots in the file OVERDUES.

4.5 The Catalog and Loans Maintenance Group

The four files in this group are NEWBOOKS, NEWENTRIES, DELETIONS, and SAVEFILE. Their interactions with each other and with the files AUTHORS, LOANS, and CATALOG during runs of the file maintenance subroutine UPDATE are shown in Figure 4.5.1.

4.5.1 The File NEWBOOKS

This MTS sequential file is a card-image file which contains the raw entries for new books that are to be added to the system's files and the raw replacement entries for old entries that are incorrect. These raw entries are prepared on decklets of punched cards which are then sorted by hand into alphabetical order according to the author-name key field that is contained in each raw entry. The sorted deck of raw entries is then read into the disk file NEWBOOKS through use of the MTS command COPY.

One raw entry, of format described below, is prepared for each author of each new book. The entries for primary

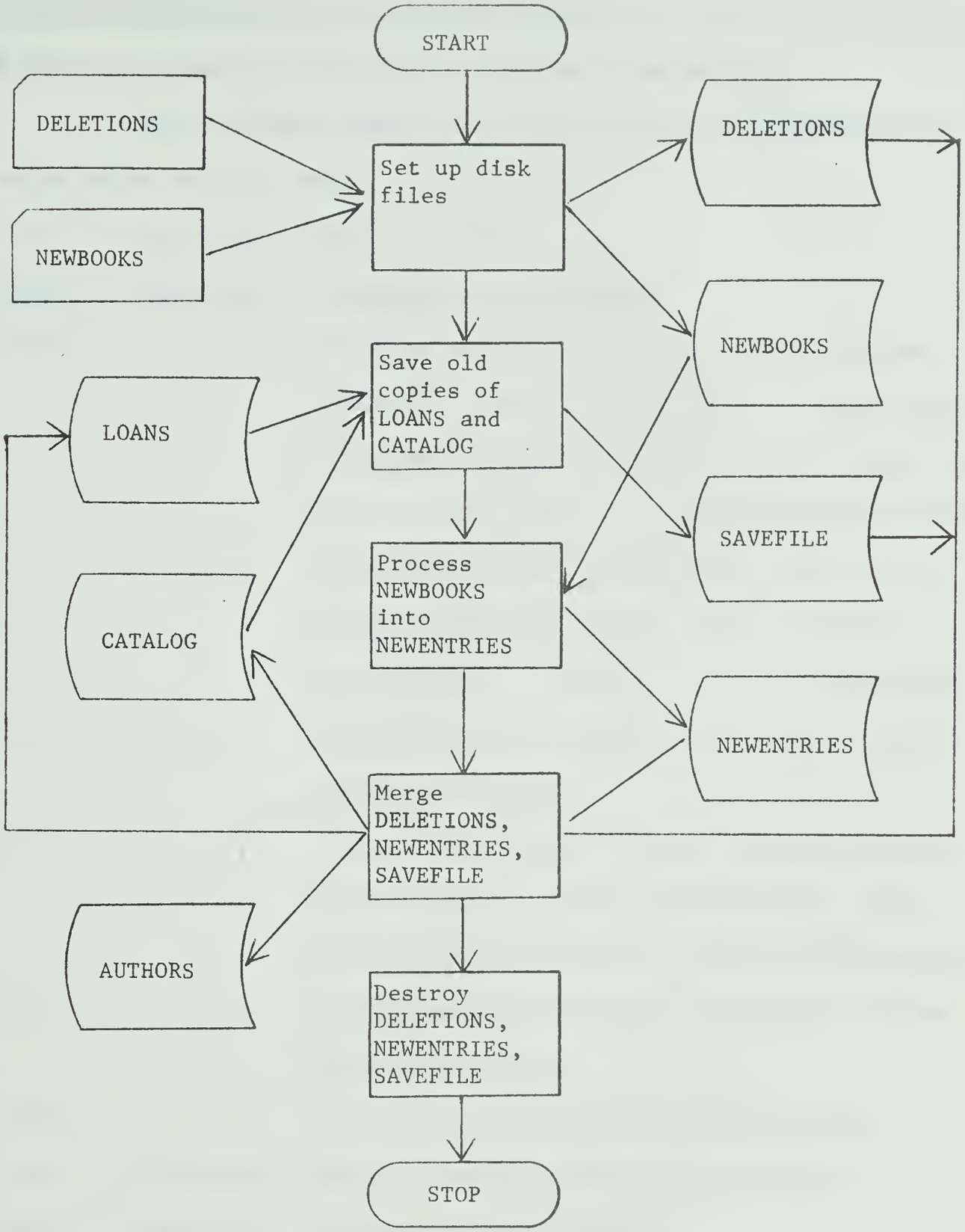


Figure 4.5.1 General Flow Chart for the Modules BATCHUPDATE and PICKUP

and secondary authors, who are defined in Section 4.2.2, are differentiated by Field No. 9 of each raw entry.

Each decklet, and hence each entry in NEWBOOKS, is structured as follows:

i) The first card or record:

Bytes	Field No.	Content of the Field
1-24	1	This 24-character field contains the author's surname, followed by one blank, followed by up to four initials, with no punctuation and no embedded blanks. The surname is left-justified within this author-name key field and is right-truncated, if need be, to 19 characters.
25-45		This field is unused at present and so contains blanks.
46-61	2	This 16-character field contains the LC call number, left-justified and right-truncated, if need be, to 16 characters.
62-74		This field is unused at present and so contains blanks.
75-80	3	This field contains serialization.

ii) The second card or record, the author card:

Bytes	Field No.	Content of the Field
1-6		This field contains blanks.
7-38	4	This 32-character field contains, in free format, the author details for the author whose name is used in Field No. 1. These author details include surname, forenames

Bytes	Field No.	Content of the Field
		or initials, dates, epithets, and relator. Relators include the terms "author," "editor," "composer," and "translator." If no relator is included in this field, then the term "author" is assumed to apply.
39-74		This field is unused at present and so contains blanks.
75-80	5	This field contains serialization.

iii) The one to eleven title cards cards or records:

Bytes	Field No.	Content of the Field
1-8		This field must contain eight blanks
9-74		This field is 66*N characters in length where N is an integer that ranges in value from 1 to 11. It contains up to nine searchable words of title, separated by single blanks, followed by two consecutive blanks which are followed by the remainder of the title, if any. The title is followed, in order, by the subtitle, if any; the complete author statement, if the book has more than one author; and the series statement, if any.
75-80		This field contains serialization.

iv) The publisher card or record:

Bytes	Field No.	Content of the Field
1-10		This field must contain ten blanks.

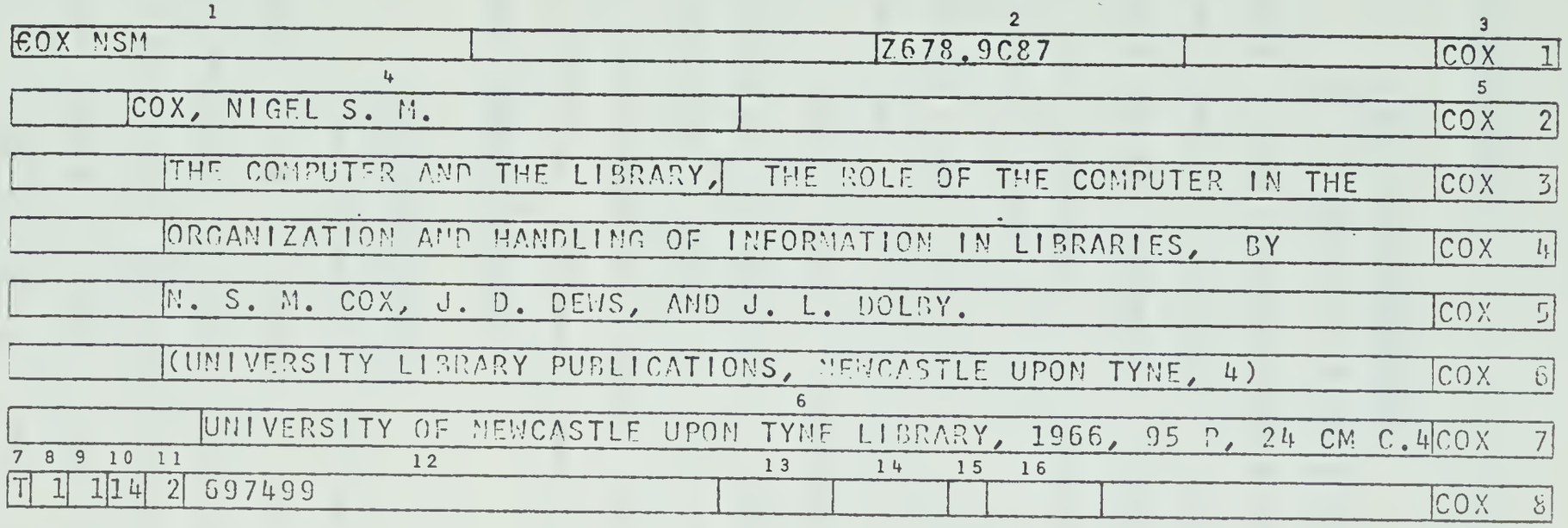
Bytes	Field No.	Content of the Field
11-74	6	This 64-character field contains the publisher, place of publication, date of publication, and the number of pages.
75-80		This field contains serialization.

v) The last card or record:

Bytes	Field No.	Content of the Field
1	7	This one-character field contains the end flag, which is always "T."
2-3	8	A two-digit code for the library that holds the book. The only code assigned at present is "01" which denotes the Computing Science Reading Room.
4-5	9	Two digits which indicate the number of copies of the book that are held by the library. This field contains blanks if the entry is for a secondary author.
6-7	10	Two digits which give the loan period of the book in days. This field contains blanks if the entry is for a reference book or for a secondary author.
8-9	11	Two digits which indicate the number of authors of the book in addition to the one whose name is given in the author-name key field. This field contains blanks if the book has only one author.
10-37	12	This 28-digit field contains four

Bytes	Field No.	Content of the Field
		seven-digit subfields that contain the accession numbers of up to four copies of the book. If these accession numbers are less than seven digits in length, then they are right-justified within their individual seven-digit fields.
38-43	13	This six-digit field contains three pairs of digits which indicate the acquisition date of the book in the form YYMMDD where the digits YY indicate the year, the digits MM indicate the month, and the digits DD indicate the day of the month.
44-49	14	This six-digit field contains the purchase order number.
50-51	15	This two-digit field contains the fund number.
52-57	16	This six-character field contains the cost of the book in the form ddd.cc where the three digits ddd indicate the number of dollars in the cost and the two digits cc indicate the number of cents in the cost.
58-74		This field is unused at present and so contains blanks.
75-80		This field contains serialization.

A sample entry from NEWBOOKS is shown in Figure 4.5.1.1. The numbers above each line, or card image, refer to the fields




```
9   DOLBY JL                                    Z678.9C87                    DOLBY1
10        DOLBY, J. L.                                                       DOLBY2
11          THE COMPUTER AND THE LIBRARY,  THE ROLE OF THE COMPUTER IN THE   DOLBY3
12          ORGANIZATION AND HANDLING OF INFORMATION IN LIBRARIES,           DOLBY4
13          BY N. S. M. COX, J. D. DEWS, AND J. L. DOLBY.                    DOLBY5
14          (UNIVERSITY LIBRARY PUBLICATIONS, NEWCASTLE UPON TYNE, 4)        DOLBY6
15            UNIVERSITY OF NEWCASTLE UPON TYNE, 1966, 95 P., 24 CM., C.4.   DOLBY7
16  T 1     2 697499                                                         DOLBY8
```

Figure 4.5.1.1 Sample Entries from the File NEWBOOKS

discussed above.

4.5.2 The File NEWENTRIES

As is shown in Figure 4.5.1, this MTS sequential file is used by the program module BATCHUPDATE when it is adding the entries for new books to the system's files. NEWENTRIES is used for the temporary storage of new catalog entries that have been generated by the subroutine DATA, which is called by subroutine UPDATE, which is described in Section 4.5.1, and processes them into entries that have the same structure as the entries in the file CATALOG, which is described in Section 4.2.1. The entries in NEWENTRIES are sequenced alphabetically according to the contents of the author-name key field that is contained in each entry and, hence, are ready to be merged with the entries in CATALOG. New entries for the files LOANS and AUTHORS are not generated until the merge step of subroutine UPDATE is run.

4.5.3 The File SAVEFILE

This MTS sequential file is used for the temporary storage of the old copies of the files CATALOG and LOANS prior to the merging, by subroutine UPDATE, of the old entries in these files with the new entries in NEWENTRIES to generate the new copies of CATALOG and LOANS as is shown in Figure 4.5.1. It is during this merge step that the new entries for the files LOANS and AUTHORS are generated by the subroutine UPDATE.

The old copy of AUTHORS is not saved because many of the pointers that it contains are no longer correct after the

new loans and catalog entries have been merged with the old loans and old catalog entries. AUTHORS is regenerated in its entirety by subroutine INDXR during the merge step of subroutine UPDATE.

The subroutine SAVE is called by UPDATE to merge the old copies of CATALOG and LOANS into SAVEFILE. Each merged entry in SAVEFILE consists of one entry from the file CATALOG followed by any corresponding entry or entries from the file LOANS. As is described in Section 4.2.2, only catalog entries for primary authors have corresponding entries in the loans file. The merging process does not alter the previously described structures of the entries from either the catalog file or the loans file. Naturally, the composite entries in SAVEFILE are sequenced alphabetically by the author-name key field.

4.5.4 The File DELETIONS

DELETIONS is an MTS sequential file in which each entry, a single line, specifies a catalog entry and its corresponding loans entry or entries, if any, that are to be deleted from the files CATALOG and LOANS during the merge step of the subroutine UPDATE.

The use of DELETIONS allows incorrect entries or entries for lost, destroyed, or retired books to be deleted from the system's files.

Each 57-byte entry in DELETIONS is structured as follows:

Bytes	Field No.	Content of the Field
1-20	1	This 20-character field contains the author surname, left-justified and right-truncated, if need be, to 19 characters.
21-24	2	This four-character field contains up to four initials, left-justified and with no punctuation or embedded blanks.
25-30		This field contains blanks.
31-46	3	This 16-character field contains the LC call number, left-justified and right-truncated, if need be, to 16 characters.
47-50		This field contains blanks.
51-57	4	This seven-digit field contains the acession number of the book. If the accession number is less than seven digits in length, then it is right-justified within this field.

A sample entry from DELETIONS is shown in Figure 4.5.4.1. The numbers above the entry refer to the fields discussed above.

4.6 The Patron Maintenance Group

The interactions of the one file in this group, NEWUSERS, with the files USERS, USERINDEX, and FREESTOR are shown in Figure 4.6.1, which applies to the off-line enrollment of new users. Figure 4.6.2, applies to the on-line enrollment of new users, which does not use the file NEWUSERS.

When it is desired to enroll new users in the batch

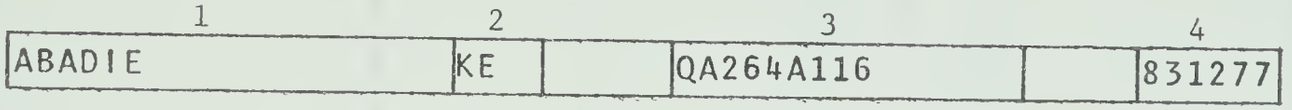


Figure 4.5.4.1 Sample Entry from the File DELETIONS

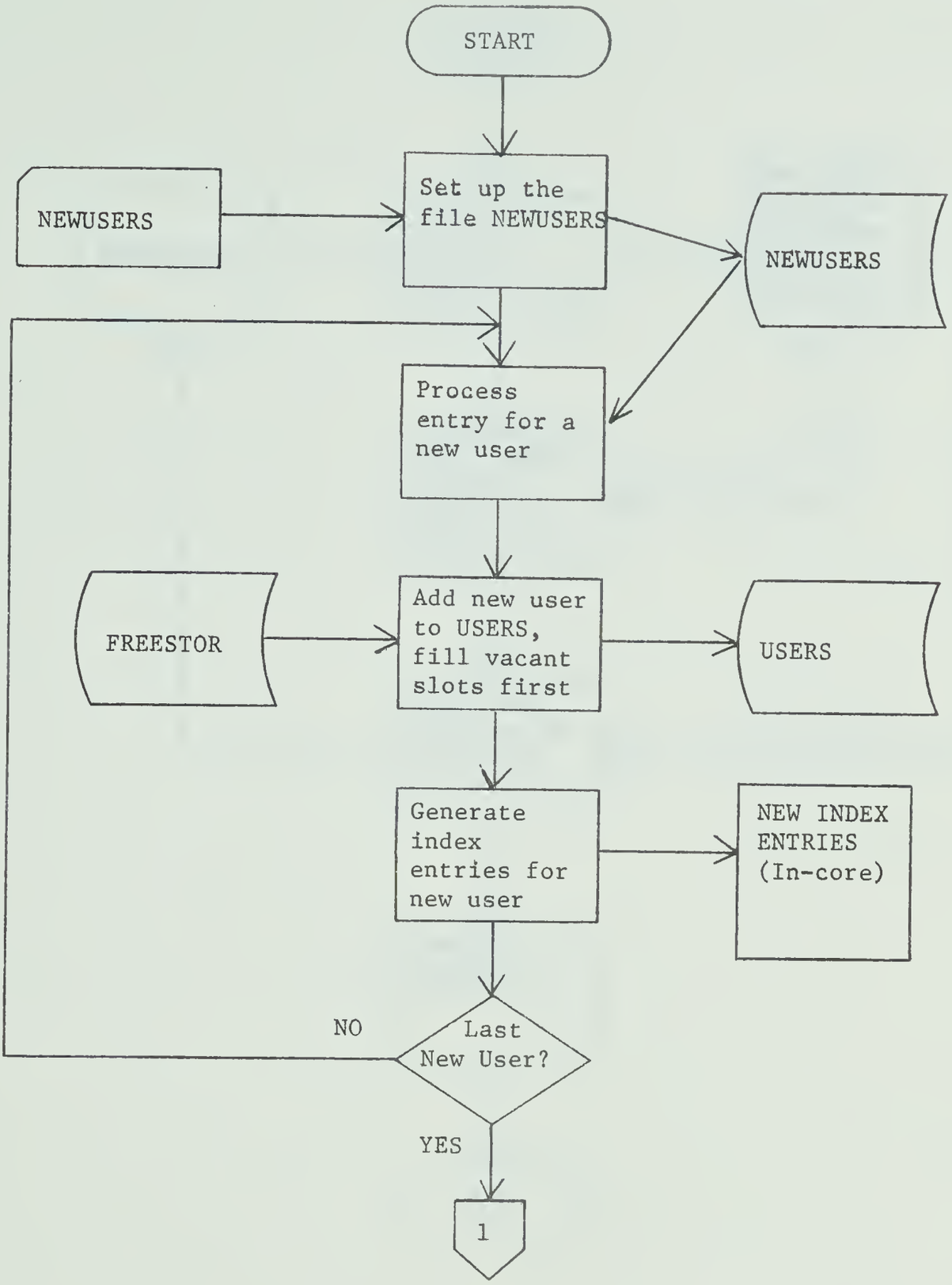
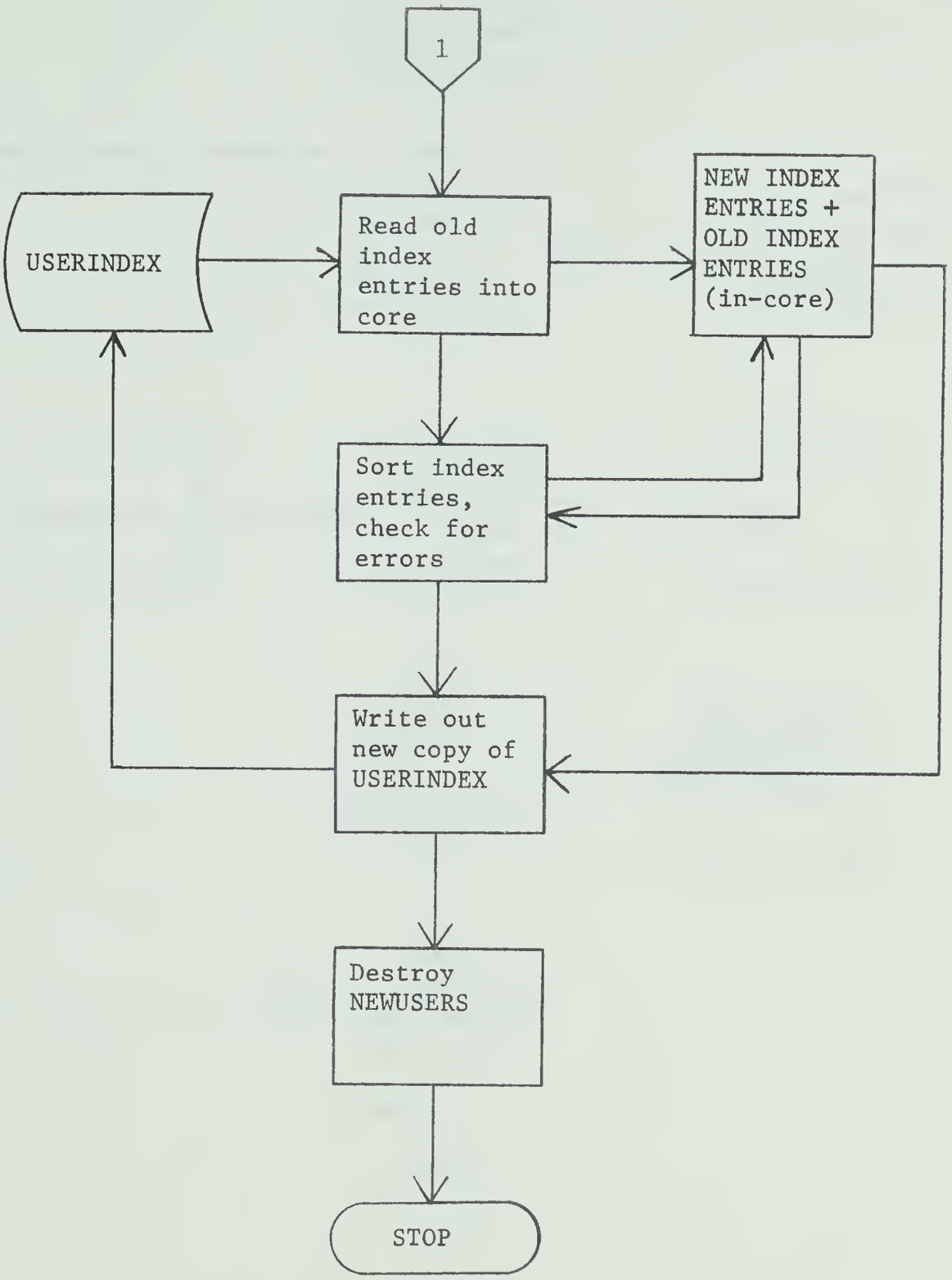


Figure 4.6.1 Off-Line Enrollment of New Users with the Batch Mode Module BATCHENROLL


1
USERINDEX
Read old index entries into core
NEW INDEX ENTRIES + OLD INDEX ENTRIES (in-core)
Sort index entries, check for errors
Write out new copy of USERINDEX
Destroy NEWUSERS
STOP

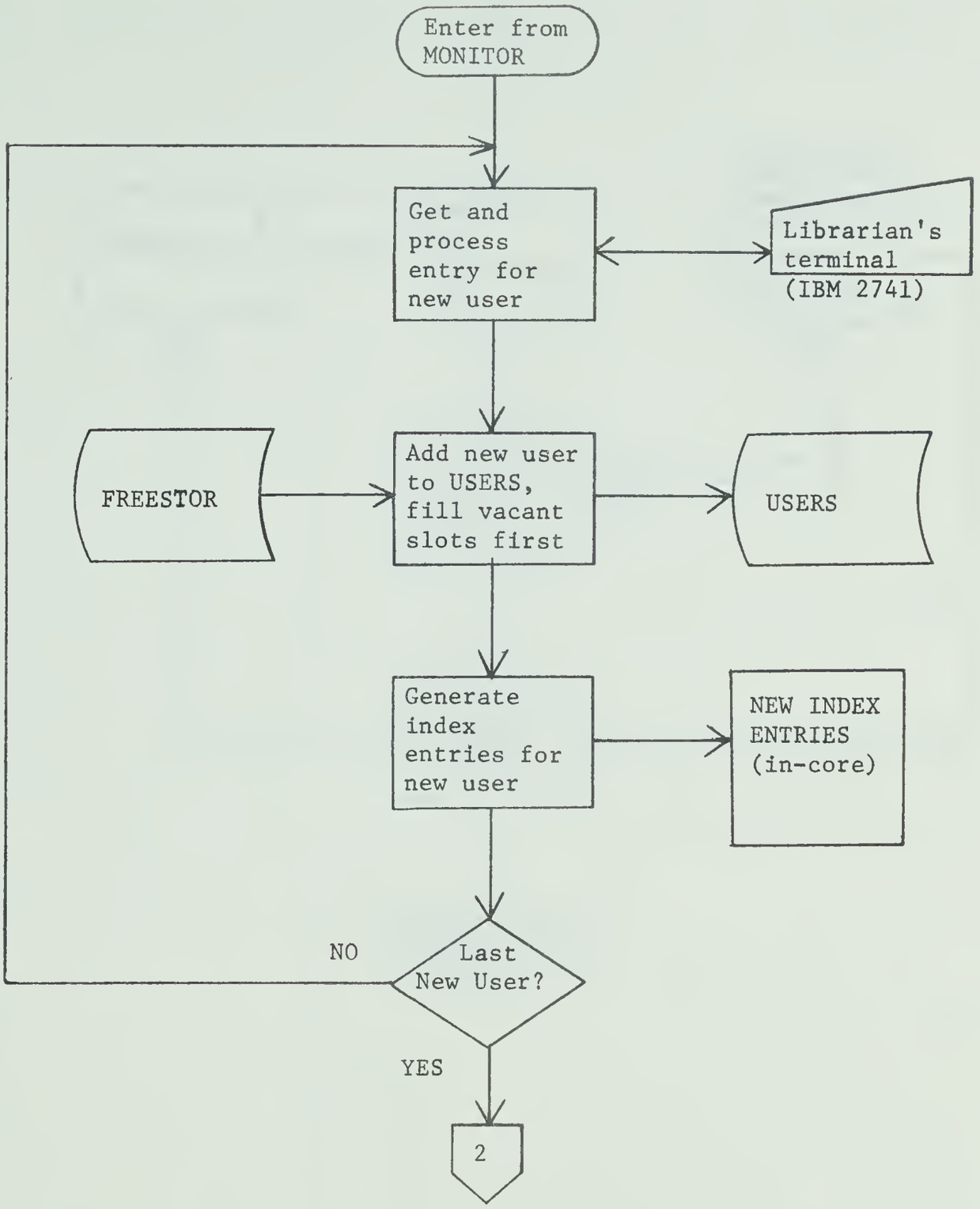
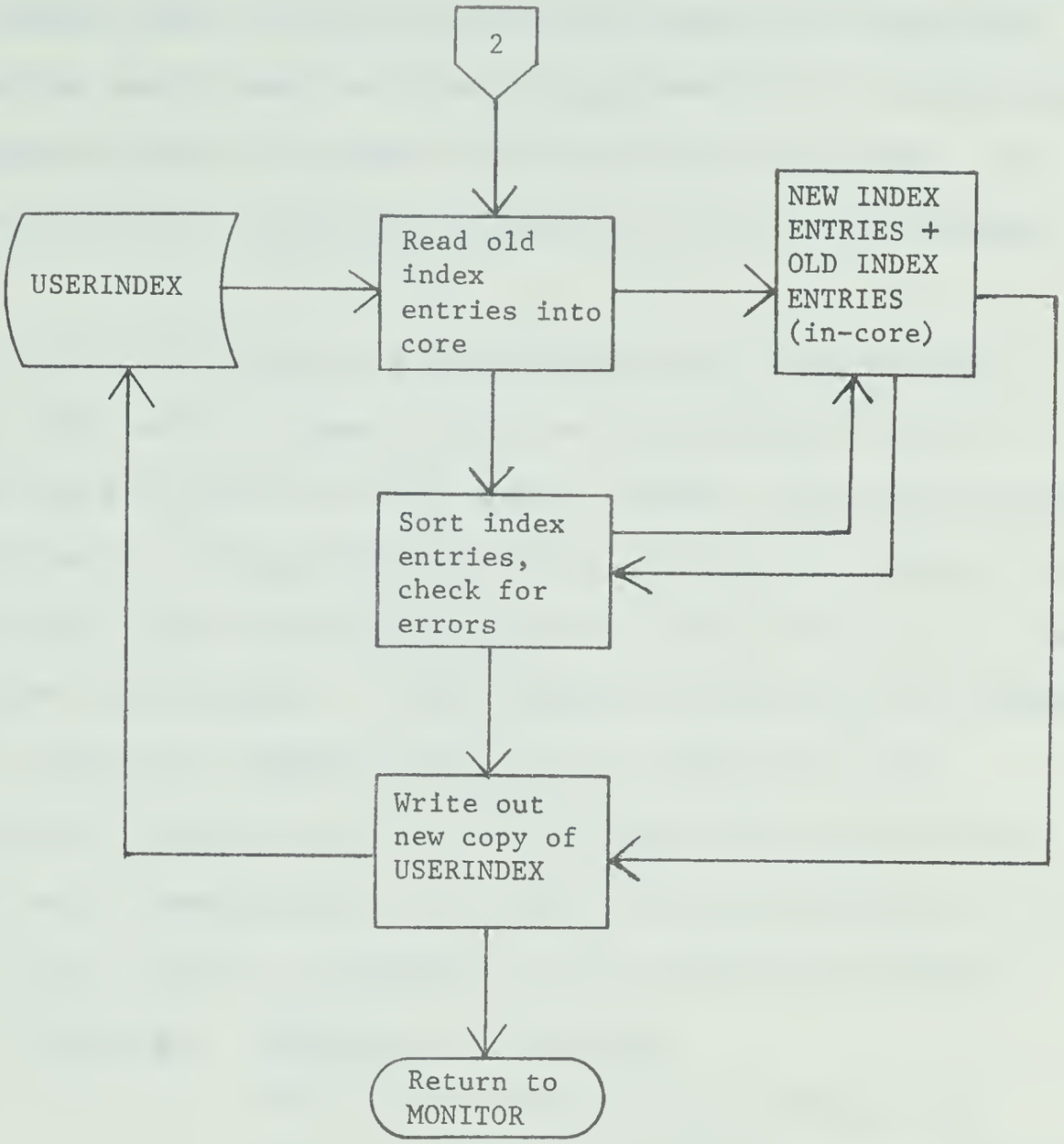


Figure 4.6.2 Enrollment of New Users from the Terminal


2
USERINDEX
Read old index entries into core
NEW INDEX ENTRIES + OLD INDEX ENTRIES (in-core)
Sort index entries, check for errors
Write out new copy of USERINDEX
Return to MONITOR

mode, rather than in the conversational mode, the required information about each new user is punched into a single card. The resulting deck of cards, which need not be sorted, is read into the file NEWUSERS through use of the MTS command COPY.

Since NEWUSERS is a card-image file, the records in this file have the same format as the punched cards. Because the enrollment of new users, whether performed on-line or off-line, is always handled by the subroutine ENROLL, this punched-card format also applies to the entry of data for new users from the terminal. When invoked by the on-line command ENROLL, subroutine ENROLL defines this format for the librarian by typing out a set of headings under which the various data elements are to be typed by the librarian.

Each record in NEWUSERS is structured as follows:

Bytes	Field No.	Content of the Field
1-16	1	This 16-character field contains the user's surname, left-justified and right-truncated, if need be, to 16 characters.
17-20	2	This four-character field contains up to four initials, left-justified, and with no punctuation or embedded blanks.
21		This field contains one blank.
22	3	This one-character field contains a one-digit status code that indicates the user's borrowing privileges. The codes used in this field are identical to those discussed, in Section 4.3.1, for Field

Bytes	Field No.	Content of the Field
		No. 3 in the records of the file USERS.
23		This field contains a blank.
24	4	This one-character field contains a one-digit code that indicates the user's title. The codes used in this field are identical to those discussed, in Section 4.3.1, for Field No. 4 in the records of the file USERS.
25-27		This field contains three blanks.
28-51	5	This 24-character field contains the address of the user, in free format.
52-58	6	This seven-character field contains the user's telephone, which is right-justified if it is less than seven characters in length.
59		This field contains one blank.
60-61	7	This field contains a two-character code that denotes the user's faculty or department. These codes are described in the discussion of Field No. 7 in Section 4.3.1.
62-71		This field contains ten blanks.
72-80	8	This nine-character field contains either a six-digit student identification number or a nine-digit social insurance number. The six-digit student identification number is right-justified within the field.

Figure 4.6.3 shows a sample entry from the file NEWUSERS. The numbers above the record correspond to the fields discussed above.

4.7 The File OFFLINE.I/O

The primary use of this MTS line file is as a system log for the on-line module LIBRARY, with every on-line transaction being logged in this file.

A secondary use of OFFLINE.I/O is for the temporary storage of overdues notices that are created during the on-line execution of subroutine DEMAND. DEMAND scans the file LOANS for overdue books and writes the overdue notices for these into OFFLINE.I/O. The temporary storage of these notices, until the on-line module LIBRARY has been shutdown, makes it possible to print them either on-line on the terminal or off-line on the high-speed line printer, whichever is judged to be more appropriate. It is very inconvenient to print a large number of these overdue notices on the low-speed typewriter terminal.

It should be noted here that a program that is in execution in the conversational mode under MTS cannot make use of the line printer, card reader, or card punch as these devices are available only to batch mode jobs. It is possible, of course, to initiate a batch job from the terminal through the use of the remote batch entry facility that is available under MTS. However, the MTS user must sign-off at the terminal before the batch job is allowed to start execution.

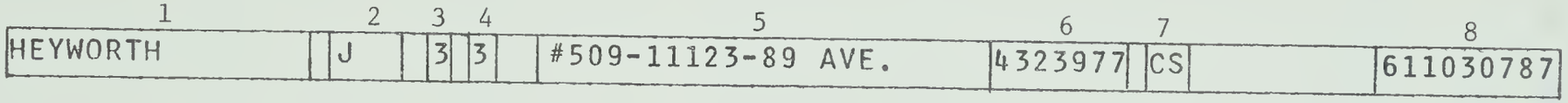


Figure 4.6.3 Sample Entry from the File NEWUSERS

By this means the librarian can cause the overdue notices to be printed off-line on the high-speed line printer while still seated at the terminal.

CHAPTER V

THE PROGRAMS OF THE IT SYSTEM

5.1 Introduction

The IT system embraces eight MTS command packages and 35 FORTRAN IV programs, each of which belongs to one or more of five modules. Table 5.1.1 lists the FORTRAN programs and indicates which of them belong to each of the modules. Similarly, Table 5.1.2 lists the MTS command packages and indicates which of them belong to the various modules. Appendix IV contains the listings, arranged in alphabetical order, for all of these command packages and programs. In addition, Appendix IV contains alphabetical indexes to the programs, the entry points within the programs, and the MTS command packages.

Two of the five modules, LIBRARY and ONLINEUPDATE, are intended to be used on-line, while the remaining three modules, BATCHUPDATE, PICKUP, and BATCHENROLL are intended to be used off-line. Each of these modules is discussed in the sections that follow.

5.2 The On-Line Module LIBRARY

As is shown in Table 5.1.1 and Table 5.1.2, the on-line module LIBRARY consists of two MTS command packages, LIBRARY and XEQ.LIBRARY, and 23 of the 35 FORTRAN programs.

LIBRARY provides all of the on-line commands discussed

TABLE 5.1.1

FORTRAN IV Programs

FORTRAN IV PROGRAMS / MODULES	LIBRARY	BATCHUPDATE	PICKUP	ONLINEUPDATE	BATCHENROLL
ALTER	X				
BINS1	X				
BINS2	X				
BINS3	X				
BLKDATA3		X	X	X	
BLKDAT4	X				X
BRING	X				
CHANGE	X				
CHECK	X			X	X
DATA		X	X	X	
DAY	X			X	X
DELET				X	
DEMAND	X				
ENROLL	X				X
ENROLL:MAIN					X
ENQUIR	X				
IDENT	X				
INDXR		X	X	X	
LBOOK	X				
LIST	X				
MAINLIB	X				
MAINUPDATE		X			
PICKUP.MAIN			X		
QUICK	X				
RESERV	X				
SAVE				X	
SAVE.BATCH		X	X		
SEARCH	X				
SETUP	X				X
SORT1	X				X
SORT2	X				X
STATUS	X				
TRNSFR		X	X	X	
UPDATE		X	X	X	
UPDATE:MAIN		X			

TABLE 5.1.2

MTS Command Packages

MTS COMMAND PACKAGES	MODULES	LIBRARY	BATCHUPDATE	PICKUP	BATCHENROLL
LIBRARY		X			
BATCHUPDATE			X		
BATCHENROLL					X
PICKUP				X	
XEQ.LIBRARY		X			
XEQ.ENROLL					X
XEQ.PICKUP				X	
XEQ.UPDATE			X		

in Sections 3.6, 3.7, and 3.8. It encompasses the real-time circulation subsystem, the on-line patron control subsystem, the on-line patron maintenance subsystem, and the on-line catalog search subsystem, which were mentioned in Section 3.1. The division of LIBRARY into these four subsystems is somewhat arbitrary since these subsystems, and the programs that comprise them, are all part of a single integrated on-line system with many of the 23 programs included in more than one of the subsystems.

Figure 5.2.1 gives an overview of the structure of LIBRARY, its subsystems, and their interactions with each other and with the system's files. The monitor and command interpreter interprets the commands that are entered from the terminal and calls the appropriate subroutines to process the transactions and queries that are specified by the commands.

5.3 The Off-Line Module BATCHUPDATE

The off-line module BATCHUPDATE is composed of two MTS command packages, BATCHUPDATE and XEQ.UPDATE, and seven of the 35 programs, as is shown in Table 5.1.1 and Table 5.1.2. BATCHUPDATE, together with the module PICKUP, comprises the off-line catalog, loans, and author-index files maintenance subsystem. This subsystem provides for the addition of entries for new books, the correction of erroneous entries already in the files, and the deletion of entries for lost, destroyed, or retired books. It may be helpful to refer to Figure 4.5.1 while reading the description that follows.

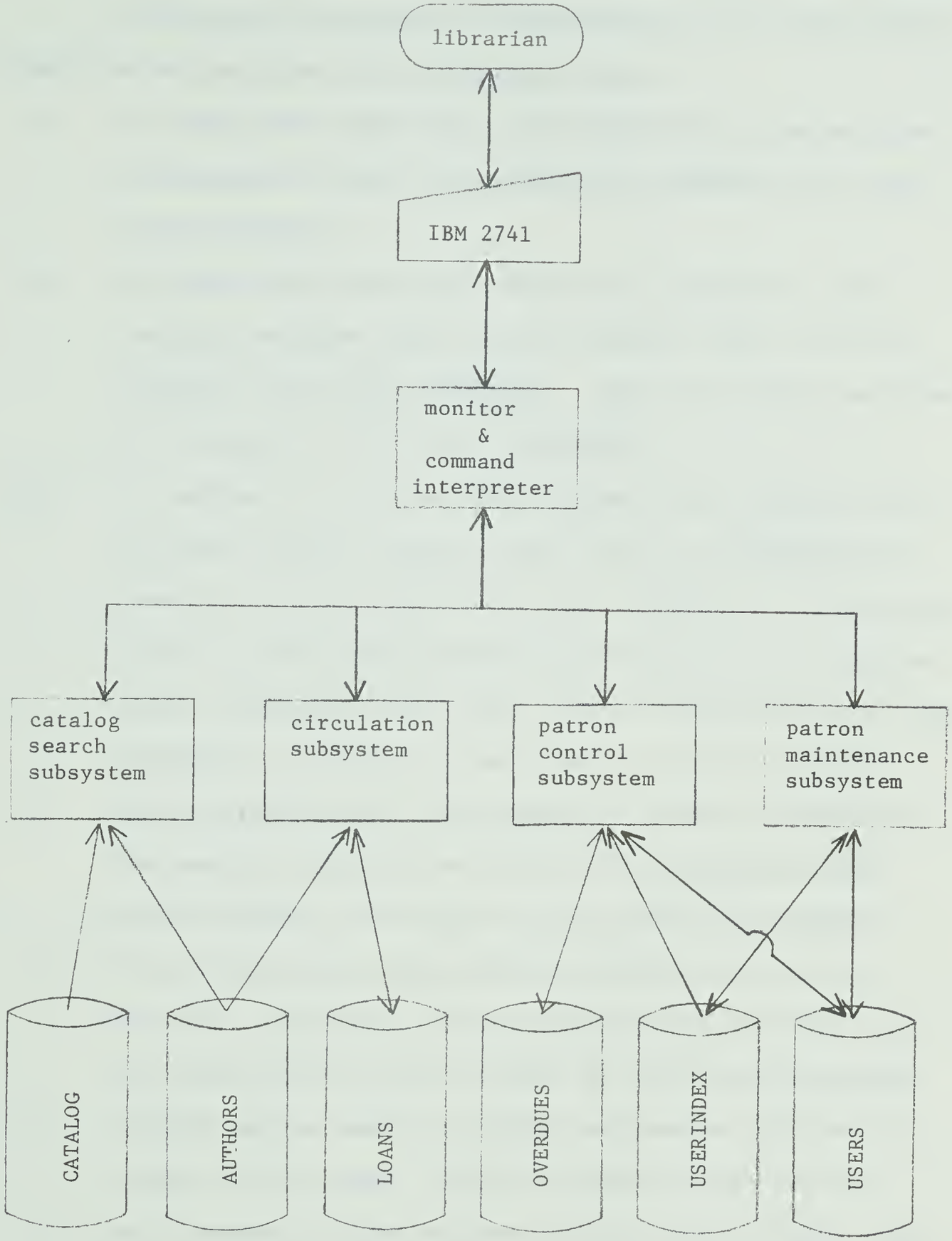


Figure 5.2.1 The General Structure of the On-Line Module LIBRARY

The central program in BATCHUPDATE is the subroutine UPDATE which performs the following tasks:

i) It calls the subroutine SAVE.BATCH which merges the old copies of the files LOANS and CATALOG into the file SAVEFILE.

ii) It calls the subroutine DATA which generates new catalog entries from the raw entries that are contained in the file NEWBOOKS. The new catalog entries are stored in the file NEWENTRIES.

iii) It merges the old catalog entries from SAVEFILE with the new catalog entries from the file NEWENTRIES to generate a new copy of the file CATALOG. At the same time it also generates new loans entries for the new books and merges them with the old loans entries from SAVEFILE to produce a new copy of the file LOANS.

iv) While building the new copies of LOANS and CATALOG, the module UPDATE also calls the subroutine INDXR which builds a new copy of the index file AUTHORS.

v) During the merge step UPDATE deletes the catalog entries, and their corresponding loans entries, that are specified by the entries in the file DELETIONS.

vi) UPDATE can be used to correct erroneous entries in either of two ways. First, if the author and the call number in the erroneous entry are correct, then UPDATE takes the first entry from the file NEWENTRIES that contains exactly that author and call number and uses it as a replacement for the old entry in the new

files that it is generating. Second, if the author or the call number is incorrect, then it is necessary to delete the erroneous entry by the inclusion of an entry for it in the file DELETIONS as well as replacing it by the inclusion of a correct entry in the file NEWBOOKS.

5.4 The Off-Line Module PICKUP

The off-line module PICKUP is used only if it is necessary to recover from an operating system crash from a crash caused by a mispunched card during the execution of BATCHUPDATE. It can perform all of the tasks performed by BATCHUPDATE, but normally will continue the update run from a point just after the last major step completed by BATCHUPDATE.

5.5 The Off-Line Module BATCHENROLL

This off-line module consists of two MTS command packages, BATCHENROLL and XEQ.ENROLL, and eight FORTRAN programs, as is shown in Table 5.1.1 and Table 5.1.2. It is the off-line counterpart of the on-line command ENROLL. Other than taking its input from the file NEWUSERS, instead of from the terminal, BATCHENROLL performs the same task as ENROLL. These tasks are outlined in Section 3.7 and are summarized in Figure 4.6.1.

5.6 The On-Line Module ONLINEUPDATE

The on-line module was used in the CP/CMS version of the IT system to carry out the maintenance of the loans,

catalog, and author-index files. It consists of ten FORTRAN programs as is shown in Table 5.1.1.

In addition to providing on-line commands that performed the same tasks as the modules BATCHUPDATE and PICKUP, this module provided three additional on-line commands, SAVE, COPY, and DISPLAY. The command SAVE allowed the librarian to save the loans and catalog files on tape, and the command COPY allowed the librarian to recopy those files back onto the disk. The command DISPLAY was used to display, on the terminal, selected portions of the loans and catalog files.

This module has not been made operational in the MTS version of the IT system because its functions can be performed more efficiently by BATCHUPDATE, PICKUP, and several MTS commands and utilities.

CHAPTER VI

PROPOSED FILE STRUCTURE FOR THE ON-LINE CATALOG SUBSYSTEM

6.1 Introduction

The remainder of this study is concerned with the design of the computer file structures that are needed to support an integrated, on-line, user-oriented library system for a large academic library. Figure 6.1 gives a highly summarized, conceptual view of such a system. The rectangles in Figure 6.1 represent those parts of the overall system that are either fully automated or semi-automated. The solid lines represent flow of information, and hence interaction, between the components of the system; and the dotted lines represent flow of physical documents between the components of the system. The technical services subsystems are charged with the performance of all functions related to the maintenance and expansion of the document collection and the files of the on-line catalog and real-time circulation subsystems. The technical services subsystems are not treated in detail in this study. The file organization of the on-line catalog subsystem is discussed in the remainder of this chapter, while the file organization of the real-time circulation subsystem is discussed in Chapter VII.

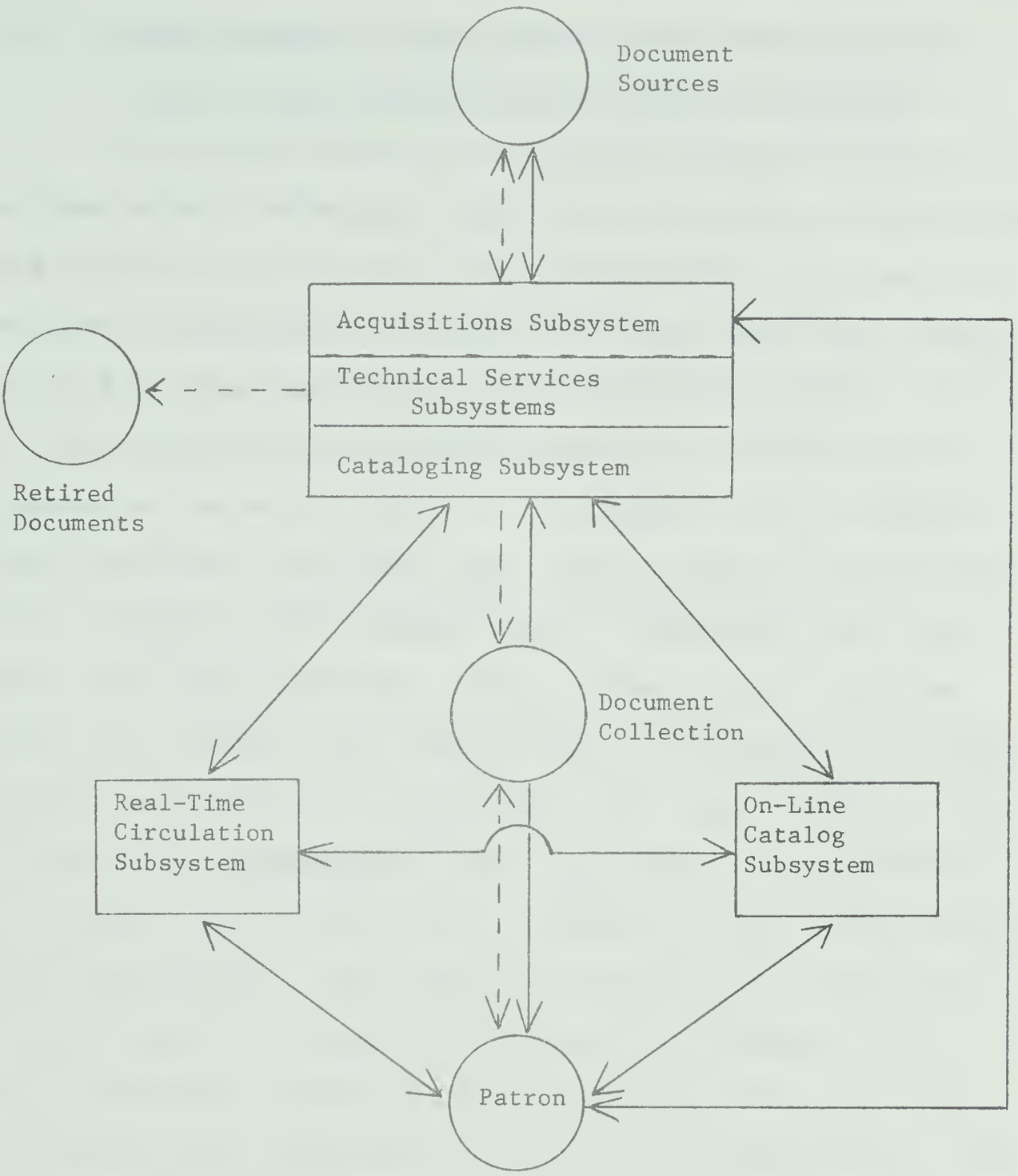


Figure 6.1 Conceptual View of an Integrated, On-Line, User-Oriented, Library System

6.1.1 Brief Review of Work Done on the Design of File Organization for the On-Line Catalog Subsystem

The basic structure of the file organizations of the on-line catalog subsystem, as conceived by most of those who have written on the topic [75,78,94,135,136], is that of a two-level organization in which the first level is a single serial file that contains the bibliographic entries of the on-line catalog and the second level is a set of inverted indexes to the serial file. I. A. Warheit [135] compared three different techniques that can be used to organize the files of the on-line catalog; and he concluded that this serial file plus inverted indexes organization, which he calls the combined file organization, is superior to either a single serial file or a threaded list organization. R. M. Curtice [137] demonstrated that for purposes of searching the single serial disk file is inferior to the inverted disk file organization. The earliest version of the IT system, a later version of which is discussed in Chapters III, IV, and V, employed a serial file organization that was found to be unacceptably inefficient in the on-line environment, even for a very small catalog. Apparently only two researchers, R. T. Divett [79] and H. P. Burnaugh [84], have attempted to apply the threaded list techniques to the organization of the files of the on-line catalog. Both the file organization developed by Divett and the file organization developed by Burnaugh are judged to be suitable for a small on-line catalog that contains 10,000, or fewer, entries; but neither file

organization is judged to be suitable for a large on-line catalog that contains 1,000,000, or more, entries. In the latter case a great many disk accesses probably would be needed to retrieve all the elements in any given list, and this in turn implies a very slow response to queries. The combined file organization has been chosen as the framework around which is built, in the remainder of this chapter, the proposed file organization for the on-line catalog subsystem.

6.1.2 Criteria for the Design of the On-Line Catalog Subsystem

The design of the file organization which supports the on-line catalog subsystem is subject to five important constraints. First, the file organization must be such that it supports queries that involve AND, OR, and NOT logic. It should also support, or at least not preclude, queries that involve the relations of ADJACENCY and PRECEDENCE and queries that involve the weighting of terms in the queries or the use of root fragments, including both right and left truncations, of terms. It is also desirable that the file organization include a thesaurus which can be used by either the user or the catalog search programs to expand, contract, normalize, or redefine the scope of the user's query. Second, the file organization and the search strategy must be such that the user of the on-line catalog system receives an answer to his query within a reasonable period of time, the

length of which varies inversely with the intricacy of the query and the load on the complete system. This load includes that caused by the circulation system which would receive high-priority scheduling for most of the transactions that it would handle. Third, the file organization, search strategy, processing programs, and scheduling algorithms must be such that the overall system can efficiently "time-share" among queries and transactions which originate from several different terminals. Fourth, since the on-line catalog is a very large file, and hence expensive to store, it is highly desirable that some method be devised to allow the file to be compressed and the entries therein to be efficiently encoded and decoded. Estimates of the size of the catalog file for one million titles range from 2×10^8 characters, which is roughly equivalent to the storage capacity of seven IBM 2316 Disk Packs, to 5×10^8 characters, which is roughly equivalent to the storage capacity of sixteen IBM 2316 Disk Packs. Considerably more important than the cost of the storage medium for such a large file is the cost of the one or two nine-drive IBM 2314 Disk Storage Devices which would be required to maintain it on-line. Fifth, given the second and third requirements it is very important to minimize the number of disk or data cell accesses required to obtain any given piece of information.

The file organization proposed below attempts to satisfy all the conditions mentioned above.

6.1.3 Overview of the File Organization of the On-Line Catalog Subsystem

The basic elements of the proposed file organization are outlined in Figure 6.1.3.1 for title words. A similar structure holds for author names, subject headings, and most other items which are used as primary access points.

The Hash Index File and the Compressed Bit-Strings File form an inverted index to the entries in the Compressed Catalog File. The Hash Index File and the Dictionary File are used to encode queries and to encode and, hence, compress new catalog entries. The Dictionary File is used to decode compressed catalog entries that satisfy queries, so that they can be displayed in a form suitable for human comsumption. Figure 6.1.3.2 shows how the basic structure would appear when a thesaurus is included. It is important to note that only one of the files shown in Figure 6.1.3.2 contains uncoded terms. That file is the Dictionary of Uncoded Terms. The Hash Index, Thesaurus, and Compressed Catalog Files contain coded pointers to this file instead of the actual terms. The Compressed Bit-Strings File contains only pointers to the Compressed Catalog. Neither the Thesaurus nor the Compressed Bit-Strings File are discussed in the present treatment. The structure of the Compressed Bit-Strings File is essentially the same as that proposed by H. S. Heaps and L. H. Thiel [112,113].

The file organization outlined above is modified somewhat for the case of LC call numbers. First, the Hash

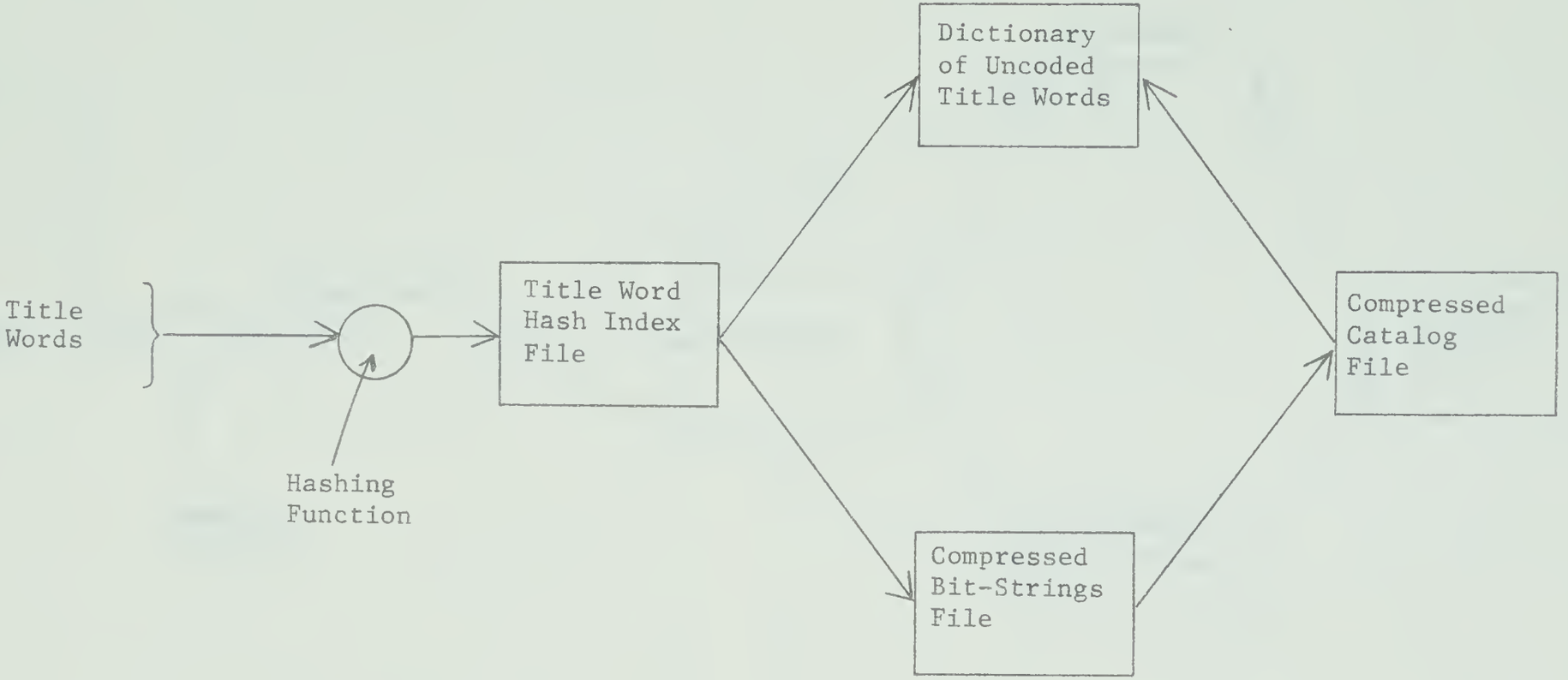


Figure 6.1.3.1 The Basic Components and Structure of the File Organization for the On-Line Catalog

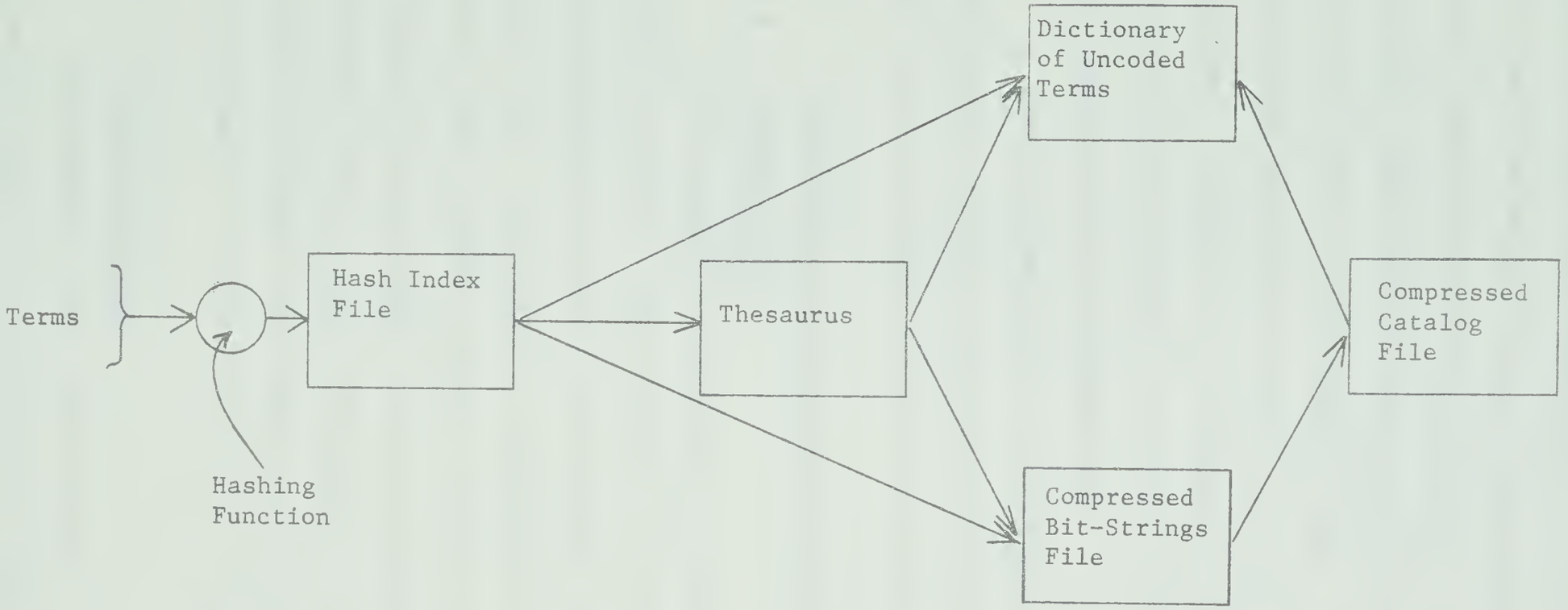


Figure 6.1.3.2 The Structure of the File Organization for the On-line Catalog When a Thesaurus Is Included

Index File and the Compressed Bit-String File are integrated into one file called the LC Call Number Index. Second, the LC Call Number File has an internal structure which is quite different from that of the other dictionary files because it, along with the LC Call Number Index, is part of the real-time circulation subsystem as well as part of the on-line catalog subsystem. Figure 6.1.3.3 gives an overview of this segment of the file organization of the combined on-line catalog and real-time circulation subsystems.

The discussion of the details of this proposed file organization begins in the next section with the theory that underlies the design of the hash index files.

6.2 The Theory of Virtual Hash Addressing

6.2.1 Introduction

The proposed structure of the hash index files, which provide the first stage indexes by title and subject words, author name, and Library of Congress call number to the catalog file, represents a synthesis of the concepts of the virtual hash address and the virtual scatter table [114, 115], the scatter index table [114], and the bucket [116]. Before describing the proposed structure of the hash index files, it is best to discuss the theory of virtual hash addressing, and this is done in the sections that follow.

6.2.2 Hash Addressing

In general, any hash storage or scatter storage scheme can be envisaged as consisting of three parts:

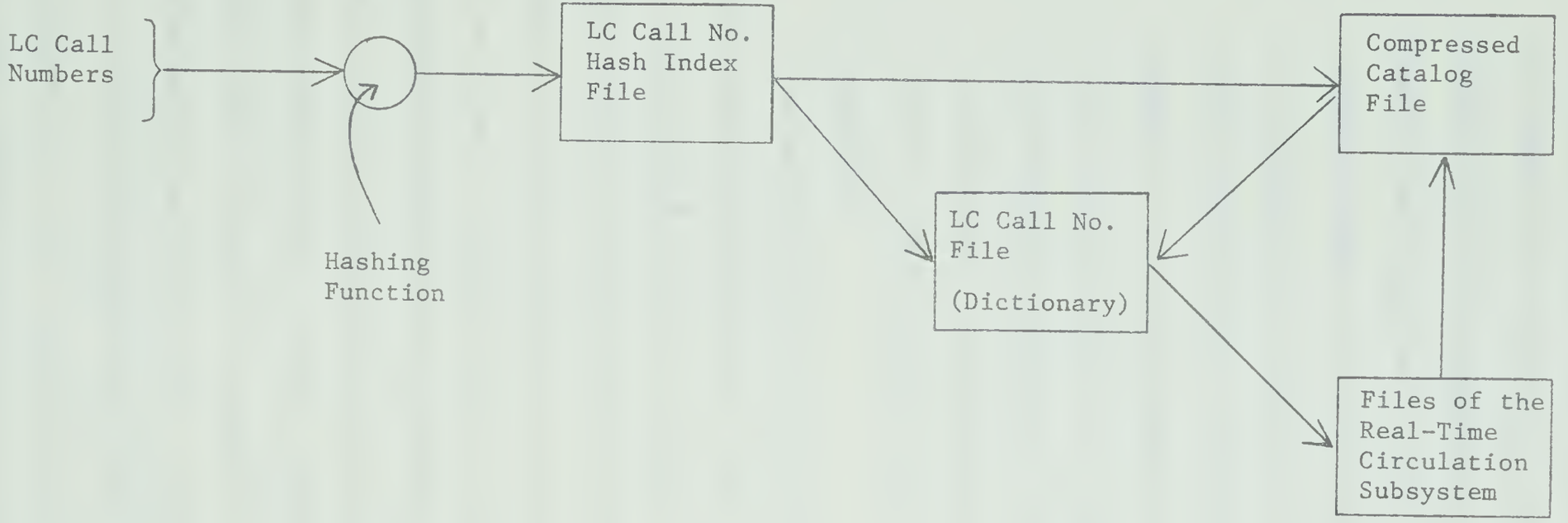


Figure 6.1.3.3 The Relationships Among the LC Call Number Hash Index File, the LC Call Number File, the Compressed Catalog File, and the Files of the Real-Time Circulation Subsystem

i) A table or file that is at least sufficiently large to store all the keys, and their associated data, that are to be mapped into it. In the present treatment a key may be a title word, an author name, a subject word, or an LC call number, among other things.

ii) A randomizing function or transformation or algorithm which operates on the keys, be they alphabetic, numeric, or alphanumeric, and generates, reproducibly, a bit string that is at least long enough to address all storage locations in the table or file.

iii) A method for resolving the collisions that occur whenever two or more keys map into the same storage location.

The exact nature of the randomizing function depends upon the nature of the set of keys to be transformed and upon their distribution; but it should, at least in theory, break up any clusters of keys and distribute the entire set of keys uniformly over the entire set of storage locations in the table or file. Various forms of the hashing or randomizing transformation have been discussed in the literature. These methods include multiplication methods [114, 116, 122], division methods [116, 122], logical methods [122], addition methods [114] truncation methods [114, 116], and radix transformation methods [116]. Since most discussions of hashing functions assume that the keys are uniformly distributed, it is of interest to note that J. Toyoda et al. [123] have derived theoretical results for the operation of a key-to-address transformation of the form (ak + b) mod C

on clustered keys, where k represents the key.

The most extensive comparison of methods for generating hash addresses appears to have been done by D. M. Murray [115] who compared the performance of nine different methods operating on two different sets of alphabetic keys, the ADI wordform which contains 7822 words and the CRAN 1400 wordform which contains 8926 words. Murray's experiments compared three variations on each of the addition, multiplication, and division methods. He concluded that the method he calls the "multiply and center" method has the best combination of collision, search, and computation properties when used to transform the words contained in the two dictionaries used in his experiments.

In the absence of further experimental results, the "multiply and center" method has been adopted as the hashing function which is used to transform the keys of the title word, subject word, author name, and LC call number indexes. This is, of course, a very risky extrapolation in the cases of author names and LC call numbers. Experiments similar to those performed by D. M. Murray should be conducted on representative samples of author names and LC call numbers to determine the optimum methods of transforming these keys into hash addresses.

6.2.3 Collisions at Real Addresses

It is highly likely that more than one key will map into any given storage location in the hash table, a situation that is known as a

collision at a real address. The two or more colliding keys are members of a collision chain.

According to D. M. Murray [115], the probability that i keys map into the same real storage location is given approximately by

$$P_i = e^{-\alpha} \alpha^i / i!, \quad i = 0, 1, 2, \ldots, N \qquad (6.2.3.1)$$

where N is the number of keys being entered into a table or file that contains H storage locations and α is the load factor equal to N/H. This probability measure is based on the assumption that the distribution of transformed keys is uniform. The probability of a collision at any particular real storage location is the same as the probability that two or more keys map into that storage location and is given by

$$\begin{aligned} P_c &= \sum_{i=2}^{N} P_i \\ &= 1 - \sum_{j=0}^{1} P_i \\ &= 1 - e^{-\alpha} (1 + \alpha) \end{aligned} \qquad (6.2.3.2)$$

Generalizing (6.2.3.2), the probability that k or more keys map into any particular real address is given by

$$P_k = \text{Prob}\ (i \geq k)$$

$$= \sum_{i=k}^{N} e^{-\alpha}\ \alpha^i/i!$$

$$= 1 - \sum_{i=0}^{k-1} e^{-\alpha}\ \alpha^i/i! \qquad (6.2.3.3)$$

The expected number of keys that map into any particular real address is, of course, α. If it is given that a collision has occurred at a particular real storage location, then the expected length of the resulting collision chain is given by

$$L = \sum_{i=2}^{N} i\ P_i/P_c$$

$$= \frac{1}{P_c} \sum_{i=2}^{N} ie^{-\alpha}\ \alpha^i/i!$$

$$= \frac{1}{P_c} \left\{ \sum_{i=0}^{N} ie^{-\alpha}\ \alpha^i/1! \ - \sum_{i=0}^{N} ie^{-\alpha}\ \alpha^i/i! \right\}$$

$$= \frac{1}{P_c} \left\{ \alpha \ - \ \alpha\ e^{-\alpha} \right\}$$

$$= \frac{\alpha(e^{\alpha} - 1)}{e^{\alpha} - (1+\alpha)} \qquad (6.2.3.4)$$

As an example, if $\alpha = 1$ then $L \simeq 2.43$

6.2.4 Virtual Hash Addressing

A virtual hash address is a hash address that contains more bits than are actually needed to address all of the storage locations in the table or file. In effect, if the

virtual hash address contains v bits and only r bits are needed to address all locations in the real scatter table, then the virtual hash address can address all locations in a virtual scatter table which is $2^{v-r} = 2^m$ times larger than the real scatter table. The group of r bits needed to address all locations in the real scatter table is called the "major" and is used to determine the storage location of the key and its associated data, if any, in the real scatter table. The remaining group of m bits is called the "minor" and is stored, along with the key and its associated data, in the table or file location pointed to by the "major".

The extra bits in the minor may be used to provide for rapid expansion of the scatter table without the need to recompute hash addresses, or they may be used to speed the search process when a key and its associated data are being retrieved. Since the minor is fixed-length and usually contains far fewer bits than the key itself, it is quicker to compare the minor of the search key against minors of keys stored in the file than to compare the search key itself against the keys stored in the file. Only if the minors match does it become necessary to compare keys. If the virtual hash address is sufficiently long, then the probability of two different keys having the same major and minor becomes infinitesimal, and the keys need never be compared or even stored in the scatter table.

These concepts were apparently first suggested by R. M. Morris [114] and later more fully developed by D. M. Murray

[115] who has proposed a virtual scatter storage scheme in which the keys are never stored in the scatter table.

6.2.5 Collisions at Virtual Addresses

Murray has developed the following approximation for the expected number of collisions at virtual hash addresses. If N is the number of keys to be mapped into storage, V is the number of virtual storage locations, and N_c is the expected number of collisions at virtual hash addresses, then

$$N_c = N^2/2V \qquad (6.2.5.1)$$

This approximation is based on the assumption that the key transformation produces a uniform distribution of virtual hash addresses, an assumption unlikely to be realized in practice.

If both sides of Equation (6.2.5.1) are divided by N then the expected relative frequency of collisions at virtual addresses, f_c, is obtained as

$$f_c = N/2V \; ; \qquad (6.2.5.2)$$

if $N = 2^n$ and $V = 2^v$ then

$$f_c = 2^{n-v-1} \qquad (6.2.5.3)$$

6.2.6 Determination of the Number of Bits Needed in the Virtual Hash Address

In a practical situation it is important to know how many bits are needed in the complete virtual hash address,

how many bits are needed in the major section of the virtual hash address, and hence how many bits are needed in the minor section of the virtual hash address. Most practical applications of virtual hash addressing are defined by three parameters:

i) N, the number of items that are to be mapped into the file scatter table.

ii) f_c, the desired frequency of collisions at virtual addresses.

iii) α, the desired load factor for the real scatter table.

The load factor is defined by $\alpha = N/H$, where H is the total number of storage locations in the real scatter table. Given these basic parameters the number of bits needed in the major, minor, and complete virtual hash address are determined as follows.

Since the major is used to address the storage locations of the real, physical file and since the number of locations needed in the real file is $H = N/\alpha$, it is readily apparent that the number of bits, m, needed in the major is the same as the number of bits needed to address all locations in the real file. Hence,

$$m = \lceil \log_2 H = \lceil \log_2 (N/\alpha) \qquad (6.2.6.1)$$

where $\lceil$ means "the smallest integer greater than or equal to". However, since most methods of generating this m-bit major produce an integer in the range 0 to $2^m - 1$, where $2^m \geq N/\alpha$, it is best to redefine H to be

$$H = 2^m \qquad (6.2.6.2)$$

where $m = \lceil \log_2 (N/\alpha)$. H and m do not depend upon f_c, but they do depend upon N and α, and through α they are dependent upon the desired frequency of collisions at real storage locations. This latter topic will not be pursued further at this time since the design of the hash index files is based on the assumption that α = constant = 1.

The number of bits, v, needed in the virtual hash address can be determined as follows. Given that $f_c = N/2V$, it is apparent that $V = N/2f_c$. Therefore,

$$v = \lceil \log_2 V = \lceil (\log_2 N - \log_2 2f_c) \qquad (6.2.6.3)$$

If it is assumed that $N = 2^n$ and $f_c = 2^{-\gamma}$, where n and γ are non-negative integers, then (6.2.6.3) becomes

$$v = n + \gamma - 1 \qquad (6.2.6.4)$$

Once the number of bits needed in the virtual hash address and in the major are known, the number of bits, p, in the minor is given by

$$p = v - m \qquad (6.2.6.5)$$

From equations (6.2.5.1) through (6.2.6.5) it is possible to determine most of the pertinent parameters of any particular application of the virtual scatter table concept. This is done later for the title word, author name, and LC call number hash indexes. Before delving into the details of

these hash indexes, it is best to describe their basic structure and to explain the mechanisms by which they are generated, maintained, and searched.

6.3 Basic Structure of the Hash Index Files

The basic structure of the hash index files, illustrated by Figures 6.3.1 and 6.3.2, is as follows. Each logical record, or bucket, in these files consists of three parts: an index section, a content section, and a counter section. The index section contains 64 one-byte pointers which normally point to entries in the content section of the bucket. The content section contains c entries each of which contains the following data elements:

i) A "minor" field of fixed-length that contains the minor of the virtual hash address of any key that is mapped into the content entry. The length of this field is always an integral number of bytes. It depends upon, among other things, the number of keys that are to be entered into each particular hash index file and the desired collision frequency at virtual hash addresses, as is indicated by Equations (6.2.6.3) and (6.2.6.5). The leading bit of this minor field is used in the resolution of collisions at virtual hash addresses, a topic that is discussed in Section 6.5.4.

ii) A one-byte field that contains a chain pointer which normally points to another content entry in the same bucket. This chain pointer field is used in the

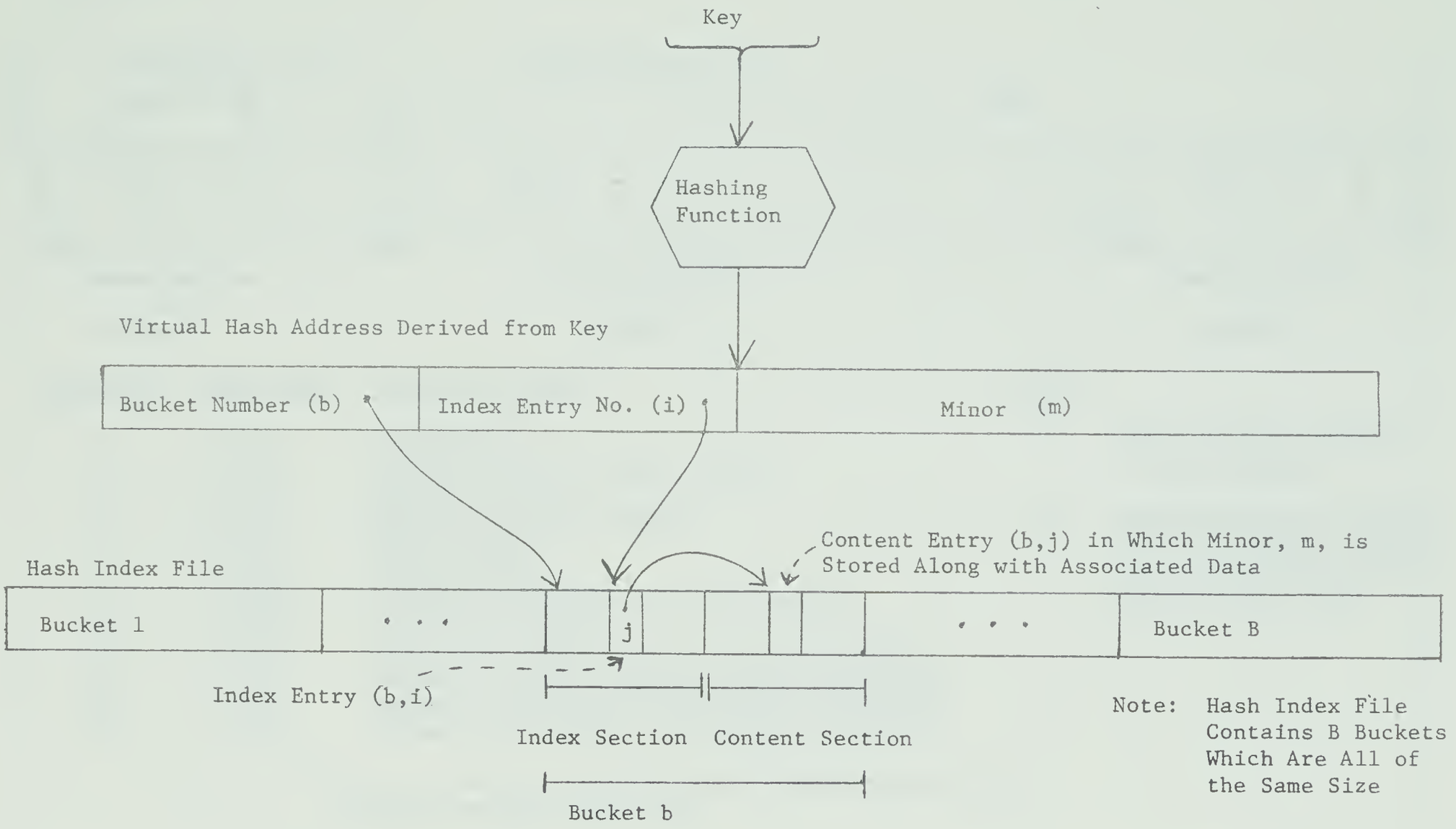
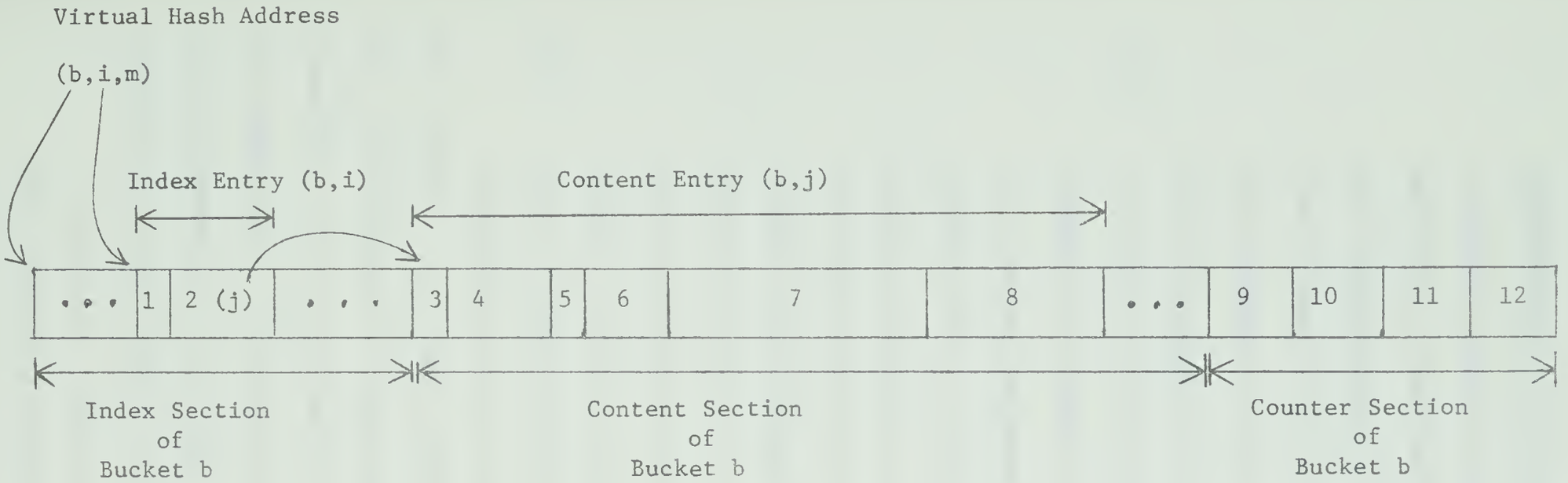


Figure 6.3.1 Basic Structure of the Hash Index File: Overview



Field No.	Field Size	Content of Field
1	1 bit	Overflow Bit
2	7 bits	Pointer to j-th content entry
3	1 bit	Collision Bit
4	r bits	Minor of Virtual Address
5	1 bit	Overflow Bit
6	7 bits	Chain Pointer
7	s bytes	Pointer to Uncoded Key in Dictionary
8	t bytes	Additional Fields
9	1 byte	Pointer to First unoccupied Content Entry
10	1 byte	Number of Index Entries Occupied (optional)
11	1 byte	Number of Overflows from Bucket (optional)
12	1 byte	Number of Overflows into Bucket (optional)

Notes:

1) Index Section contains 64 Index Entries.

2) Content Section contains C Content Entries, where C is a constant for all buckets in the file.

* r, s, t are integers and have fixed values for any particular hash index file

Figure 6.3.2 Basic Structure of the Hash Index File: Details of Buckets

resolution of collisions at real addresses in the file, a topic that is discussed in Section 6.5.3. In addition, the leading bit of this chain pointer and the leading bit of each entry in the index section of the bucket are used in the handling of overflows from the bucket, a topic that is discussed in Sections 6.5.5 through 6.5.7.

iii) A fixed-length field that contains a pointer to the character representation, stored in a dictionary file, of the key represented by the content entry. The length of this dictionary pointer field is always an integral number of bytes. It depends upon the number of keys that are to be mapped into each particular index file and upon the structure of the dictionary file that contains the uncoded keys. This dictionary pointer also serves as a code for the uncoded key, which is not stored in the hash index file.

iv) One or more fixed-length fields, the nature of which depends upon the particular file in question.

The number c of these content entries per bucket is a constant for all buckets in any particular hash index file. Consequently, all of the buckets in a particular hash index are the same size. This allows the direct calculation of the disk address of any bucket from its number. The counter section of each bucket contains four one-byte counters:

i) A counter, called the vacancy pointer, that points to the first unoccupied entry in the content section of

the bucket. The value of this counter is always one plus the number of occupied content entries. Whenever a new key is inserted into the bucket, it is installed in the content entry pointed to by this vacancy pointer, which is then incremented by one.

ii) A counter that indicates the number of occupied entries in the index section of the bucket.

iii) A counter that indicates the number of overflows from the bucket.

iv) A counter that indicates the number of overflows into the bucket.

The last three counters are optional. However, they are quite useful for monitoring the performance of the hashing function and the performance of the routines that handle bucket overflows.

6.4 Theoretical Considerations Related to the Choice of Bucket Capacity

The exact number, c, of entries in the content section must be less than or equal to 127 because only seven bits are available in each entry of the index section to address them and zero is reserved to indicate an unused index entry. The value of c is chosen so that the probability of overflow of any given bucket is reduced to 0.01 or less, without undue waste of space.

Using results given in [117, 118, 119], the following approximation to the probability that a particular bucket will overflow can be readily derived. If b denotes the number of index entries per bucket, c denotes the number of content entries per bucket, N denotes the total number of index entries in the entire file, n denotes the number of keys to be mapped into the file, α denotes the load factor which is equal to n/N, and P_i denotes the probability that the ith bucket will overflow, then P_i is given by

$$P_i = \sum_{k=c+1}^{\infty} (\alpha b)^k \frac{e^{-\alpha b}}{k!} \qquad (6.4.1)$$

Using tables of the Poisson distribution in [120] and [121], Table 6.4.1 was constructed. It gives the values of c and the corresponding values of the ratio c/b for selected

TABLE 6.4.1

Values of c and c/b for Selected Values of b

Subject to the Conditions that

$P_i \leq 0.01$ and $\alpha = 1$

b	c	c/b
1	5	5.00
2	6	3.00
3	8	2.66
4	10	2.50
5	11	2.20
6	13	2.17
7	14	2.00
8	15	1.88
9	17	1.89
10	18	1.80
11	19	1.73
12	20	1.67
13	22	1.69
14	23	1.64
15	24	1.60
16	25	1.56
17	27	1.59
18	28	1.55
19	29	1.53
20	30	1.50
60	80	1.33
100	125	1.25

values of b subject to the conditions that $P_i \leq 0.01$ and $\alpha = 1.0$. It is clear from the table that the greater the value of b, the smaller need be the value of c/b to maintain $P_i \leq 0.01$. W. Buchholz [116] gives a similar table for the cases of c = constant = b and α varying from 0.1 to 1.2 in steps of 0.1. This table indicates that P_i decreases as b increases, given a fixed value of α. Clearly, it is advantageous in terms of storage cost to use as large a value of b as is practical; hence, the above choice of b=64 in Section 6.3.

If b = 64 and the hash index file is expected to be fully loaded, $\alpha = 1$, then the best value of c is 1.33 x 64, or approximately 85. Lesser values of c are appropriate for a file expected to be more lightly loaded, $\alpha < 1$, and greater values of c are appropriate for a file expected to be more heavily loaded, $\alpha > 1$, since as indicated by Table 6.4.1 and the table given in [247] the function $c = f(\alpha, b)$ can be very roughly characterized by $c = \frac{k\alpha}{b}$. Table 6.4.1 could be easily extended to include values of other than $\alpha = 1$.

The above discussion leads to the conclusion that the proposed scheme retains most of the advantages of the usual scatter index method, in which the index entries and content entries are kept in two separate files [114], while providing the additional advantage that a second disk access is not normally required to obtain the content entries which are pointed to by the index entries. This saving in access time is, of course, bought at the expense of an increase in storage cost. For example, if b = 64, $\alpha = 1$, and $P_i \leq 0.01$, then

c = 85 and c/b = 1.33. This implies that 33% more space must be allocated for content entries than if the two files were kept separate. Of course, the requirement that $P_i \leq 0.01$ can be relaxed to $P_i \leq 0.05$ or $P_i \leq 0.10$ with a consequent reduction in the ratio c/b, but this would be attended by considerably more bucket overflows, and hence extra disk accesses.

6.5 Insertion of Keys into the Hash Index Files

6.5.1 Introduction

Now that the appropriate groundwork has been laid in the preceding sections, it is appropriate to return to the discussion of the basic structure of the hash index files and in particular to the method used to insert keys into these files, the method used to search these files for keys, the method used to resolve collisions at both real and virtual addresses, and the methods used to handle bucket overflows.

6.5.2 The Simplest Case of Key Insertion

Whenever an item or key, denoted by K, is to be entered into, or searched for in, one of the hash index files, its character representation is transformed into a v-bit virtual hash address, H(K), as is indicated in Figure 6.3.1. This virtual address is then broken up into three segments:

i) A b-bit segment, B(K), which specifies the bucket into which the key maps.

ii) An i-bit segment, I(K), which specifies the index entry within the bucket.

iii) A [v-(b+1)] - bit segment, M(K), which is the minor of the virtual hash address.

When the bucket specified by B(K) has been read into core, the index entry specified by I(K) is examined to determine whether or not it is empty. The index entry will contain all zero bits if it is empty. If the index entry is empty, then the key has not been previously entered into the file. In this case the bucket's vacancy pointer is examined to determine whether or not there is a vacant entry in the content section of the bucket.

If there is a vacant entry left in the content section, then the new item is placed in it, the current value of the vacancy pointer is inserted into the index entry specified by I(K), and the vacancy pointer is incremented so that it points to the next vacant entry, if any, in the content section. The minor of the virtual hash address is inserted into the minor field of the content entry and any data associated with the key is inserted into the appropriate fields of the content entry. Additionally, the chain pointer field is set to zero to indicate that the newly entered item or key is the last item or key on the newly formed chain which originated in the index entry specified by I(K), and the dictionary pointer is updated so as to point to the character representation of the key or item which is stored in a separate dictionary file as is indicated by Figure 6.3.2. In general, the dictionary pointer contains within itself a one-, two-, or three-byte code which is used to replace the key or item

in the bibliographic entries of the compressed catalog. The structures of these pointers and codes and the structures of the dictionaries are discussed in later sections.

6.5.3 The Method Used To Handle Collisions at Real Addresses

The previous section describes the insertion of keys when only one key has mapped into a given index entry, or real storage location. However, there is a relatively high probability, approximately 0.37 from Equation 6.2.3.2, that two or more keys will map into the same index entry, or real address, when $\alpha = 1$. Hence, some method of handling these collisions at real storage locations must be available.

The method used with the proposed hash index files is as follows. If a key maps into a nonempty entry in the index section of the bucket specified by the bucket-segment of the virtual hash address, then the contents of the index entry are examined further to determine whether or not the overflow bit is on. (See Figure 6.3.2.) If the overflow bit is on then the remaining seven bits of the index entry are a pointer into the contents section of another bucket. The leading bit of the chain pointer field in each content entry is also an overflow bit and it serves the same purposes as does the overflow bit in index entries. This topic is pursued further in a later section that deals specifically with bucket overflows.

If the overflow bit is off, then the remaining seven bits of the index entry point to the first member of a chain of entries in the content section of the same bucket. In this case, the minor of each entry in the chain is compared, in turn, to the minor segment of the virtual hash address of the key which is being inserted until either a match is found or the chain is exhausted.

If no match is found then the new key is inserted as described above, but with the exception that the pointer to the new entry is inserted into the chain pointer field of the entry which was previously the terminal member of the collision chain.

If a match is found then either the key has been previously entered or two different keys have mapped into the same virtual address. If collisions at virtual addresses are being checked for, then the dictionary entry for the old key, the one which is already in the index, is read from the appropriate dictionary file and the new key, the one being inserted, is compared with this dictionary entry. If they are not identical, then a collision has occurred at a virtual hash address and it is handled as described in the next section. This sequence of events is dependent on the assumption that whenever new keys are encountered they are immediately added to the indexes. However, if the construction of the indexes is a "one-shot" proposition and the input list of keys is known to contain no duplicates, then it is unnecessary to read the dictionary entry to verify that a

collision has occurred at a virtual address.

6.5.4 The Method Used to Handle Collisions at Virtual Addresses

In addition to collisions at real storage addresses which occur relatively frequently, there are the above mentioned collisions at virtual addresses which, by design, occur extremely rarely. Collisions at virtual addresses are handled as follows.

If k keys collide at a virtual hash address then they are stored in k consecutive positions in the chain that hangs from the index entry into which they, and all other members of the chain, were mapped by the hashing function. The k consecutive chain entries form a virtual address collision chain. The collision bit, see Figure 6.3.2, in each of the first k-1 of these entries is turned on to indicate that each of these k-1 entries is in collision, at the same virtual address, with the entry which immediately follows it on the chain. If necessary the chain is reordered by changing the chain pointers to get the virtual-address-collision entries into consecutive positions on the chain.

6.5.5 The Methods Used to Handle Bucket Overflows

In the preceding sections the discussion touched upon the overflow bit that is contained in each entry in both the index and content sections of every bucket. It was mentioned that the overflow bit, if "on", indicated that the rest of the pointer field pointed to a content entry

within another bucket and not to a content entry within the same bucket as is normally the case.

If an item or key which is to be inserted into a hash index file maps into a bucket in which all of the content entries are already occupied by keys that have previously mapped into that bucket, then that bucket is said to have overflowed. By design the probability of this occurrence is very low, ≤ 0.01, but even so some method of obtaining space for the new key must be available.

Methods of obtaining the required space are of five types:

i) Rewrite the entire hash index file and increase the number of content entries in each bucket. This method cannot be used when the number of content entries per bucket is already 127 because the pointer fields contained in the index entries and content entries are only seven bits in length. It is not practical to increase the number of content entries in any particular bucket without uniformly increasing the number of content entries in all the other buckets of the file because this would make impossible direct calculation of the disk address of any particular bucket. In any event it is likely that the entire file would have to be rewritten to expand any particular bucket.

ii) Rewrite the entire file and split each bucket into two new buckets each of which contains the same number

of index entries as the old bucket and as many content entries as are needed to reduce the probability of any particular bucket overflowing to 0.01 or less. This doubles the number of real storage locations, or index entries, contained in the file without doubling the size of the overall file. Incidentally, this also halves the load factor α. Each key contained in the old bucket is assigned to either the first or the second of these new buckets depending upon the leading bit of either its minor, M(K), or its index address, I(K). The effect is to shift one bit from either M(K), or I(K) to the bucket address, B(K). This method is based on a suggestion by R. Morris [114].

If the leading bit of I(K) is shifted to B(K) then it is necessary to left-shift by one bit position the entire contents of I(K) and insert the leading bit of M(K) into the vacated trailing bit position of I(K). If this is not done and the leading bit of I(K) is left undisturbed, an exact replica of it being concatenated to B(K), then those keys that have a "0" leading bit in I(K) will always map into the upper half of the index section of the first bucket of the new pair of buckets and those keys that have a "1" leading bit in I(K) will always map into the lower half of the index section of the second bucket.

This may not be entirely undesirable because it would allow the old bucket to be split into a pair of new buckets each of which contains only half as many

index entries as the old bucket and up to 127 content entries. Call this method ii-b and the previously described method ii-a. Thus the content entry capacity of the file could be doubled without changing the total number of index entries in the file. However, according to the principle expounded in Section 6.4, reduction of the number, b, of index entries per bucket means that a higher value of the ratio c/b, where c is the number of content entries per bucket, is required to reduce the probability of any particular bucket overflowing to 0.01 or less. Hence, it is likely that the overall amount of space needed for the complete file will be greater than if method ii-a were used.

The circumstances under which method ii-b would require less space can be determined as follows. Let c be the number of content entries needed per bucket if method ii-a is used and let c' be the number of content entries per bucket if method ii-b is used. Let m be the number bytes needed per content entry. Then the total number of bytes needed for each pair of new buckets is b + 2c'm if method ii-b is used and 2b + 2cm if method ii-a is used. Consequently method ii-b requires less space if and only if

$$b + 2c'm < 2b + 2cm$$

or

$$c'/b - c/b < 1/2m \qquad (6.5.5.1)$$

iii) Use some algorithm to find another bucket which has space available and insert the new key into it. Algorithms for determining a sequence of buckets which can be examined for free space are discussed in Section 6.5.6.

iv) Displace a key from the overflowed bucket and place it into an overflow bucket, and then insert the new key into the vacated space of the original bucket. The overflow bucket can be found by the same method used in iii) above. The mechanics of displacing a key are discussed in Section 6.5.8.

v) Rehash all the keys into a new file which contains twice as many index entries as the old file and as many content entries as are needed to reduce the probability of any given bucket overflowing to 0.01 or less.

Each of the five methods described above can be used depending upon circumstances. The fifth method is the most costly of the five and is to be avoided except as a last resort. The second method should not be used unless the utility of the first method has been exhausted; that is, it should not be used unless each bucket in the file already contains 127 content entries. The first method, in turn, should be used only when the file is nearly saturated; that

is, when most buckets in the file are full or nearly full. The third method is the method of choice so long as a large proportion of all the buckets in the file still contain available space. The fourth method, involving the displacement of a key previously entered into a bucket, is reserved for a special case which will be described in a later section.

6.5.6 Algorithms for the Selection of Overflow Buckets

6.5.6.1 Criteria

The method chosen for the task of selection of overflow buckets must satisfy the criteria which follow:

i) It must not destroy the positional information inherent in the major segment of the virtual hash address. That is, the information provided by B(K), the bucket address, and I(K), the index address, must not be lost since it is needed to ensure that entries for overflowed keys may be found by the search program.

ii) Since the chain of entries for keys that map into any given index entry can have one and only one pointer to a content entry in an overflow bucket, all overflows that map into a given index entry must map into the same sequence of overflow buckets.

iii) The method should require little or no additional file storage space.

iv) The method should not require excessive computation.

v) It should not be prone to primary clustering.

vi) Not all of the overflows from any particular bucket should map into the same sequence of overflow buckets. Moreover, it is desirable that overflows that map into different index entries within a bucket should map into different overflow buckets because this reduces secondary clustering.

vii) The method chosen should require a minimum of extra disk accesses.

Several methods are examined in the next section.

6.5.6.2 Possible Algorithms

There are five general methods available:

i) The linear search method and its variants [114, 115, 116, 117, 118, 122, 124].

ii) The quadratic residue method [122, 125].

iii) The quadratic quotient method [126, 127].

iv) The random probe method [114, 115].

v) The chaining method [114, 115, 116].

In the linear method, if bucket i has overflowed an examination is made of buckets $(i + cj) \pmod n$, where $j \in \{1,2,3, \ldots\}$, c is a constant integer greater than zero, and n is the number of buckets in the file. The buckets are examined, in turn, until either the key has been found, a bucket with available space has been found, or the search has returned to the original bucket. This method tends to produce primary clustering [114, 122, 127] for small values of c, and

this increases the probability that bucket j(> i) will overflow given that bucket i has overflowed. The principal advantage of the linear method is that a key which maps into a full bucket is likely to be stored in a bucket which is close to the original bucket, provided that the load factor, α , is less than 0.80 [114, 118].

This method is rejected for three reasons:

i) In a time-sharing environment it is unlikely that the search program could retain the exclusive control of the disk access mechanism that is essential if the access time is to be minimized by examination of overflow buckets in sequence.

ii) If α is large then primary clustering is likely to cause the search program to search many buckets before the correct one is found.

iii) Most importantly, the method does not satisfy the first criterion and, therefore, cannot be used in its traditional forms unless the complete virtual hash address, rather than just the minor, of every transformed key is stored in the file. This would lead to a significant increase in storage costs.

A variation of the linear method involving chaining can be used with no additional storage cost. The scheme described in Section 6.3 would be modified to the extent that pointers to content entries would occupy a full eight bits and would range in value from 1 to 255. They could then

address content entries in several consecutive buckets as follows. If c is the number of content entries per bucket and p is the value of the pointer, then the displacement from the original bucket, say bucket i, of the bucket that contains the overflow entry is the integer quotient of p/c; the required content entry within bucket i + ⌊(p/c) is the remainder of p/c, that is p[mod c], plus one. This modified linear method intensifies the problem of primary clustering because it severely restricts the number of buckets into which any given bucket can overflow. Since the largest possible bucket displacement is ⌊(255/c), the total number of overflow buckets available to any given bucket is the integer quotient of 255/c. In severe cases in which all buckets in the overflow range are full, it is necessary to displace entries that originally mapped into their present location from the overflow range in order to insert a new overflow. This leads to a significant increase in the mean number of disk accesses required to retrieve keys.

The chaining method usually involves the placement of overflows into a separate overflow area which is chained to the primary area. Since this form of the method would involve extra storage costs for the separate overflow area, it is rejected for use in the present circumstances.

All of the remaining methods, the quadratic residue method, the quadratic quotient method, and the random probe method, have the virtue that they tend to reduce primary clustering [127]; and all, when used with ordinary in-core

scatter tables, require about the same number of probes for given values of α, the load factor [114, 127]. However, the quadratic quotient method fails to satisfy the second criterion because displacements from the initial hash address depend upon the key being hashed, specifically upon the integer quotient of K/N where K is the key and N is the number of locations in the scatter table. This, in turn, means that different keys mapping into the same index entry would likely map into different sequences of overflow buckets. Neither the quadratic residue method nor the random probe method suffers from this defect; but both are rejected, at least in the forms reported in the literature, for use in the present circumstances because they do not satisfy the sixth criterion.

6.5.6.3 The Algorithm Chosen

A method that does satisfy all of the aforementioned criteria is the following variant of the quadratic residue method. If a key, K, initially maps into bucket $b_o(K)$ and into index entry $i(K)$ within that bucket, then a sequence of possible buckets, in which K may be stored, is addressed by

$$b_j(K) = (b_o(K) + (i(K) + 1)\, j^2)[\text{mod } B] \qquad (6.5.6.3.1)$$

where $b_j(K)$ is the bucket number of the jth bucket in the sequence, $j \in \{0,1,2,3, \ldots, \lfloor[(B+1)/2]\}$, and B is the number of buckets in the file. Buckets addressed by $b_j(K)$ for $j > 0$ are called overflow buckets. The usual forms of the quadratic

residue method [122, 127] can be defined by the sequence

$$b_j(K) = (b_o(K) + aj + bj^2) \text{ [mod B]}$$

where a and b are constants. In the variation presented here, a has been set to zero and b has been made variable, specifically a function of i(K).

6.5.7 Insertion of Keys When Bucket Overflows Are Taken Into Consideration

If a key, K, being inserted into the hash index file initially maps into bucket $b_o(K)$ and into index entry i(K) within that bucket, then the sequence of events is as follows:

i) The chain of content entries hanging from the index entry i(K) is followed to its terminal member. If one or more members of the set of keys that map into i(K) have previously overflowed bucket $b_o(K)$, then it is necessary to follow the chain of entries into one or more overflow buckets, which are determined by Equation (6.5.6.3.1), to find the terminal member. If the terminal entry is the kth entry in bucket $b_j(K)$ where $b_j(K)$ is defined by Equation (6.5.6.3.1), then it is denoted by e(j, k) for convenience.

ii) If bucket $b_j(K)$ has space available, then the minor of the key, M(K), and its associated data are installed as previously explained in Section 6.5.2.

iii) If bucket $b_j(K)$ is full, then the key must be stored in the next bucket in the sequence defined by Equation

(6.5.6.3.1). Therefore, bucket $b_{j+1}(K)$ is examined to determine if it has any free space.

iv) If bucket $b_{j+1}(K)$ has space available, then the minor of the key, M(K) and its associated data are installed in that bucket's first available content entry, denoted by $e(j+1, c_{j+1})$, where c_{j+1} is the pointer to the first available content entry in bucket $b_{j+1}(K)$. Then the chain pointer field of e(j, k) is set equal to $c_{j+1} + 128$. Adding 128 turns on the overflow bit.

v) If bucket $b_{j+1}(K)$ has no available space, then space must be acquired by displacing a key from bucket $b_{j+1}(K)$ or by using one of the other methods mentioned in Section 6.5.5. The overflowed key must be stored in bucket $b_{j+1}(K)$ and not in one of the buckets addressed by $b_{j+\ell}(K)$, $\ell > 1$; otherwise the chain will be broken and the first criterion violated.

6.5.8 The Displacement Algorithm

An algorithm that can be used to displace keys is as follows:

Step 1. Scan the index section of the bucket until a non-empty index entry is found or until all index entries have been examined. If all index entries have been examined, then it is not possible to displace a key from the bucket and another method of acquiring space must be used.

Step 2. If a non-empty index entry is found, then examine

its overflow bit. If the overflow bit is "on", then continue with Step 1. Otherwise, go to Step 3.

Step 3. Follow the chain hanging from the non-empty index entry to its terminal member, checking the overflow bit in each chain pointer. If any overflow bit is "on", then continue with Step 1. Otherwise, go to Step 4.

Step 4. Move the terminal member of the chain to its first overflow bucket which is determined from Equation (6.5.6.3.1) and the $b_o(K)$ and $i(K)$ associated with the displaced key.

Step 5. If the first overflow bucket is full, then use this displacement algorithm to displace a key from it.

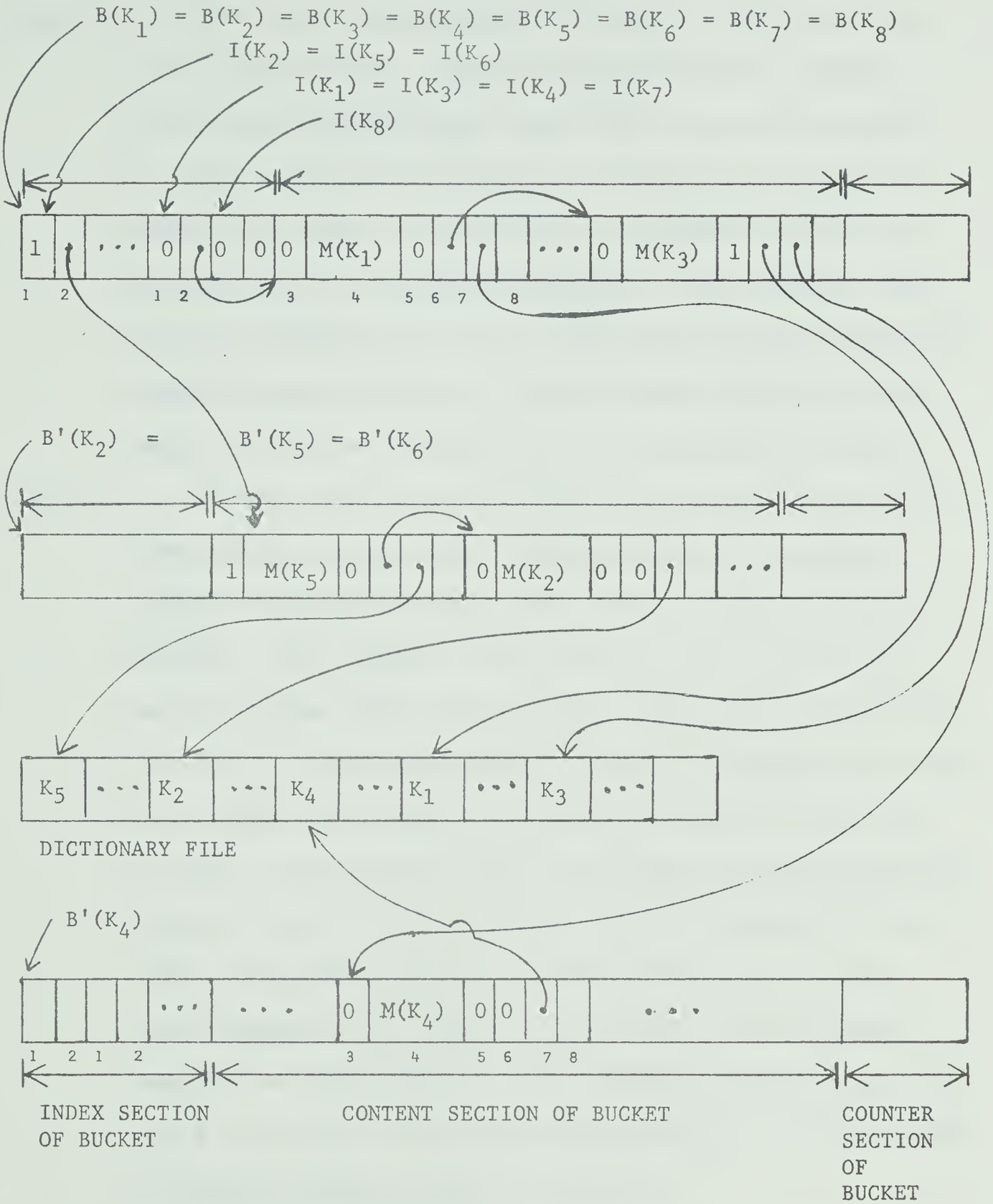
6.6 The Retrieval of Keys and Associated Data From the Hash Index Files

6.6.1 The Search Algorithm

While reading the explanation of this algorithm it may be helpful to refer to Figures 6.3.1 and 6.3.2 as well as Figure 6.6.1.1. The algorithm used to search for keys in the hash index files is as follows:

Step 1. The search key's virtual hash address, H(K), is generated in the manner described above in Section 6.5.2.

Step 2. The bucket specified by the bucket address, B(K), is read into core.



Note: The Numbers Below the Fields Refer to the Numbered Fields Shown in Figure 6.3.2.

Figure 6.6.1.1 Searching the Hash Index Files

Step 3. The index entry specified by the index address, I(K), is examined. If the index entry is empty, that is, it contains all zero bits, then the search key has not been entered previously into the hash index file and a "search key not found" result is returned to the calling program. For example, the key K_8 in Figure 6.6.1.1 is a key that has not yet been entered into the file. If the index entry is non-empty, then its overflow bit is examined in Step 4.

Step 4. If the overflow bit is "on", then the remaining seven bits of the index entry point to a content entry in another bucket that acts as the overflow bucket. For example, the keys K_2, K_5, and K_6 all map into the index entry $I(K_2)$ which has its overflow bit "on". These three keys were all entered into the file after the content section of bucket $B(K_2)$ was filled. In this instance the search program goes to Step 5. On the other hand, if the overflow bit is "off", then the remaining seven bits point to the first member of a chain of entries in the content section of the bucket. For example, keys K_1, K_3, K_4, and K_7 all map into index entry $I(K_1)$. In this event the search program goes to Step 7.

Step 5. The bucket address, B'(K), of the overflow bucket is generated by means of the algorithm described in Section 6.5.6.3. For example, the keys K_2, K_5, and K_6 all map into the overflow bucket $B'(K_2)$ and the

key K_4 maps into the overflow bucket $B'(K_4)$.

Step 6. The overflow bucket is read into core and the search program proceeds to Step 7.

Step 7. The minor of each entry in the chain of content entries is compared, in turn, to the minor of the search key's virtual hash address, M(K), until either a match is found or the chain is exhausted. If no match is found then the search key has not been entered previously into the file and a "search key not found" result is returned to the calling program. For example, the minors, $M(K_6)$ and $M(K_7)$, of the search keys K_6 and K_7 cannot be matched because these two keys are not in the file. The leading bit of the chain pointer in each content entry is an overflow bit which is checked before the search program attempts to move to the next entry in the chain. If this overflow bit is "on" then the chain of content entries is continued in an overflow bucket. For example, the key K_4 was entered into the file after the content section of bucket $B(K_4)$ was filled. It was, therefore, overflowed to bucket $B'(K_4)$ which had space available in its content section. In this instance the search program reverts to Step 5.

Step 8. If a match is found for M(K) then that entry's collision bit is examined. If the collision bit is "on" then the search program goes to Step 9. If the collision bit is "off", then the content entry which

corresponds to the search key has probably been found, and the appropriate data fields are returned to the calling program. There is a slight residual probability that an error has occurred because a misspelled search key, or a search key not yet entered into the file, may map into the same virtual hash address as some other key which is in the file. This mismatch problem is considered further in Section 6.6.2. If absolute accuracy is required, then the dictionary entry which corresponds to the content entry is read into core and compared with the search key.

Step 9. The dictionary entries that correspond to the content entries in the virtual address collision chain are, in turn, read into core and compared with the search key until either a match is found or the set of dictionary entries is exhausted. For example, $M(K_2) = M(K_5)$ and so K_2 and K_5 are in collision at the virtual hash address $H(K_2)$. Therefore, it is necessary to compare K_2 and K_5 with the dictionary entries pointed to by the content entries that contain $M(K_2)$ and $M(K_5)$. The collision bit of the content entry that contains $M(K_2)$ is "off" because it is the last member of the virtual address collision chain. If a match is found then the search key's content entry has been found and the appropriate data fields of that content entry are passed back to the calling program.

Otherwise, a "search key not found" result is returned to the calling program.

It should be pointed out to the reader that only rarely will the search program need to execute all of the nine steps outlined above. This is because the hash index files are designed so that the probability of executing Steps 5 and 6 is reduced to about 0.01 and the probability of executing Step 9 is reduced to about 2^{-16}. An additional consequence of this design is that the expected number of disk accesses required to find a key is approximately 1.01.

6.6.2 The Mismatch Problem

The mismatch problem referred to in Step 8 above is not as serious a difficulty as might be initially supposed. If it is assumed that the key transformation uniformly distributes all keys, including new and misspelled ones, over the entire virtual hash address space then the probability that a new or misspelled key maps into a virtual hash address already occupied by some other key is

$$P_c = N/V \qquad (6.6.2.1)$$

where V is the total number of addresses in the virtual hash address space and N is the number of keys that have been entered into the file. Therefore, the probability of any particular search key being mismatched is

$$P_m = P_s \times P_c \qquad (6.6.2.2)$$

where P_s is the probability that the search key is a new key or a misspelled key.

If the search keys are assumed to be title words then the order of magnitude of P_m can be ascertained as follows. If it is assumed that $N = 2^{16}$ and $V = 2^{31}$, then $P_c = 2^{16}/_{2}31 = 2^{-15} \simeq 0.00003$. Hence, $P_m \simeq 0.00003P_s$. If it is further assumed that $P_s = 0.1$, then $P_m \simeq 3 \times 10^{-6}$.

Because P_m is extremely small, the extra disk accesses needed to obtain the dictionary entries in Step 6 of the search algorithm are not worthwhile unless the collision bit is "on". In addition, if any particular new or misspelled search keys cause mismatches frequently enough to be a serious annoyance then these offending keys can be entered into the hash index file. In fact, the query system can provide a certain amount of automatic spelling correction if common misspellings of frequently misspelled keys are entered into the hash index files and equated with the correct spellings of the keys. This concept may also be extended to include correct, but variant, spellings of keys.

6.7 Frequency Loading of Keys

If the frequency of occurrence in queries distribution is known for a set of keys that are to be entered into a hash index file, then the expected search time for keys in the set can be reduced by loading the keys in order of their expected frequency of occurrence [116, 119]. Those keys with the highest expected frequency of use are loaded first and so are placed at or near the heads of the content entry chains that

hang from the index entries in the hash index file. Conversely, those keys with the lowest expected frequency of use are loaded last and so are placed at or near the ends of the content entry chains. In addition, if any keys overflow their home buckets, then they are likely to be the infrequently used keys which are loaded last. Consequently, the mean search time for keys in the set is reduced in two ways. First, the expected number of buckets that must be accessed to find a search key is reduced; and second, the expected amount of CPU time needed to find a search key, once its bucket has been read into core memory, is reduced.

6.8 The Title Word Hash Index File

6.8.1 Estimate of the Number of Different Title Words in One Million Titles

Before the detailed structure of any hash index file can be designed, it is necessary to have at least an estimate of the number of keys that are to be entered into it. An estimate of the number of different title words that are likely to be found in a collection of one million titles is obtained as follows.

If D represents the total number of documents in the library collection, N Represents the total number of words found in the titles of the D documents, W represents the total number of different title words found in the titles of the D documents, and $\bar{T}$ represents the mean number of words per title; then the following relationships may be assumed:

i) $\bar{T} = 5.5$ (6.8.1.1)

ii) $N = \bar{T}D = 5.5D$ (6.8.1.2)

iii) $\text{Log } W = 0.6 \log N + 1.2$ (6.8.1.3)

These relationships are based on statistics gathered by W. D. Reid [128] from the MARC Tapes issued over the period March 1969 to May 1970. This set of MARC Tapes contained the catalog records for approximately 57,800 documents.

If $D = 1.0 \times 10^6$ then Equations (6.8.1.2) and (6.8.1.3) yield $N = 5.5 \times 10^6$ and $W = 1.8 \times 10^5$.

6.8.2 Determination of the Number of Bits Needed in the Major, Minor, and Complete Virtual Hash Address

The number, m, of bits needed in the major of the virtual hash address depends upon the number of keys that are to be entered into the hash index file and upon the desired load factor, α. If $N = 1.8 \times 10^5$ and $\alpha = 1$ then Equation (6.6.4) yields $m = \lceil \log_2 (1.8 \times 10^5) = 18$ bits. The total number, v, of bits needed in the virtual hash address depends upon the desired frequency, f_c, of collisions at virtual hash addresses. If $f_c = 2^{-16}$, then Equation (6.2.6.6) yields $v = \lceil [\log_2 (1.8 \times 10^5) - \log_2 2^{-16} - 1] = 33$ bits. Consequently, the number of bits needed in the minor is given by Equation (6.6.8) as $p = 33 - 18 = 15$ bits.

With the above choice of $v = 33$ bits, Equation (6.2.5.1) yields a value of approximately 2 for the expected

number of collisions at virtual hash addresses. D. M. Murray [115] derived the following approximation for the probability that the number, C, of collisions at virtual hash addresses lies in the interval [a, b]:

$$\text{Prob}[a \leq C \leq b] = \sum_{i=a}^{b} e^{-\beta} \beta^i / i!, \quad i \in \{0,1,2, \ldots, \lceil \tfrac{1}{2}N \rceil\} \qquad (6.8.2.1)$$

where β is the expected number of collisions at virtual hash addresses, and is defined by Equation (6.2.5.1). If $\beta = 2$ then $\text{Prob}[0 \leq C \leq 8] = 0.9998$. Hence, the above choices for the values of m, v, and p should endow the title word hash index file with excellent virtual address collision properties.

6.8.3 Structure of the Title Word Hash Index File

All of the hash index files have the same basic structure, which is shown in Figures 6.3.1 and 6.3.2; they differ only in the structures of their content entries. Therefore, it is sufficient to describe the structure of the content entries in any hash index file to define that file's structure.

Each content entry in the title word hash index file is ten bytes in length and is composed of the following data elements:

i) A two-byte virtual address minor field of which the leading bit is the virtual address collision bit.

ii) A one-byte chain pointer field of which the leading bit is the overflow bit.

iii) A three-byte pointer to an entry in the title word dictionary file. The structure of this pointer is described in Section 6.12.2.

iv) A four-byte pointer to an entry in the title word compressed bit strings file. This pointer is replaced by the entry from the bit strings file if that entry is only four bytes long.

6.8.4 Number of Content Entries Needed in Each Bucket

The number , C, of content entries needed in each bucket to reduce to 0.01 the probability that any given bucket will overflow is determined with the aid of Equation (6.4.1). If 1.8×10^5 keys are mapped into 2^{18}, $\simeq 2.6 \times 10^5$, real storage locations then $\alpha \simeq 0.7$. If $b = 64$ and $\alpha = 0.7$ are substituted into Equation (6.4.1) then the required number of content entries is found to be $c = 62$.

6.8.5 Bucket Size

In the previous section it was found that each bucket requires 62 content entries. Hence, each bucket is composed of the following data elements:

i) 64 one-byte index entries as is shown in Figure 6.3.2.

ii) 62 ten-byte content entries.

iii) Four one-byte counters which are described in Figure 6.3.2.

Therefore, each bucket requires 688 bytes of storage space.

6.8.6 Size of the Title Word Hash Index File

Since according to Section 6.8.2 the complete index file must contain 2^{18} entries and since each bucket contains 2^6 index entries, the file must contain $2^{18} \div 2^6 = 2^{12}$ buckets. Therefore, the complete title word hash index file requires 2^{12} buckets x 688 bytes/bucket = 2,818,048 bytes of storage. The track capacity table in IBM [110] indicates that nine of these buckets fit into one track of an IBM 2316 Disk Pack. Therefore, the complete file requires $2^{12} \div 9 =$ 455.1 tracks or 11.38% of an IBM 2316 Disk Pack. If the buckets are blocked at two per physical record then the corresponding figures are 10 buckets per track, 409.6 tracks, and 10.24%. If the buckets are blocked into one-track physical records then the corresponding figures are 10.6 buckets per track, 386.4 tracks, and 9.66%. It must be remembered that these space savings achieved by blocking are partially, perhaps wholly, offset by increased buffer sizes, record transmission times, and access times. The optimum blocking factor depends upon the environment in which the system is to operate and so no attempt is made to determine it in the present treatment.

6.8.7 Expected Time to Access a Bucket

If it is assumed that the disk and the channel are idle, then the expected time to access a bucket is the sum of the expected seek time, the expected rotational delay, and the record transmission time. If an IBM 2314 disk unit is used

then the expected access time for unblocked buckets is 60ms + 12.5ms + 2.8ms = 75.3ms. The corresponding expected access times for the two cases of blocked buckets discussed above are 77.5ms and 97.5ms.

6.9 The LC Call Number Hash Index File

6.9.1 Number of Bits Needed in the Major, Minor, and Complete Virtual Hash Address

If the library collection contains 1.0×10^6 titles then 1.0×10^6 LC call numbers must be stored in the LC call number hash index file. If $N = 1.0 \times 10^6$ and $\alpha = 1$ then Equation (6.2.6.4) yields $m = \lceil \log_2 (1.0 \times 10^6) = 20$ bits. If $f_c = 2^{-16}$ then Equation (6.2.6.6) yields $v = \lceil [\log_2 (1.0 \times 10^6) - \log_2 2^{-16} - 1] = 35$ bits. It follows from Equation (6.2.6.8) that $p = 35 - 20 = 15$ bits.

The expected number of collisions at virtual addresses is given by Equation (6.2.5.1) as $\beta = N_c = 16$. Therefore, according to Equation (6.8.2.1) it follows that Prob $[0 \leq C \leq 34] = 0.9998$.

6.9.2 Structure of the LC Call Number Hash Index

As is explained in Section 6.8.3, it is sufficient to describe the structure of the content entries in any hash index file to define that file's structure.

Each content entry in the LC call number hash index is composed of the following data elements:

i) A two-byte virtual address minor field of which the leading bit is the virtual address collision bit.

ii) A one-byte chain pointer field of which the leading bit is the overflow bit.

iii) A three-byte pointer to the LC call number file which is described in Section 7.2.2. This pointer serves as a code for the LC call number that maps into this content entry.

iv) A three-byte pointer to the appropriate entry in the compressed catalog file.

Therefore, each content entry requires nine bytes of storage.

6.9.3 Number of Content Entries Needed in Each Bucket

If 1.0×10^6 keys are hashed into 2^{20} locations then $\alpha \simeq 1.0$. As was determined earlier in Section 6.4, if $b = 64$ and $\alpha = 1.0$ then the required number of content entries is $c = 85$.

6.9.4 Bucket Size

In the previous section it was found that each bucket requires 85 content entries. Consequently, each bucket is composed of the following data elements:

i) 64 one-byte index entries as is shown in Figure 6.3.2.

ii) 85 nine-byte content entries.

iii) Four one-byte counters which are described in Figure 6.3.2.

Therefore, each bucket requires 833 bytes of storage.

6.9.5 Size of the LC Call Number Hash Index File

Since the complete index file must contain 2^{20} index entries as was determined in Section 6.9.1, and since each bucket contains 2^6 index entries, the file must contain $2^{20} \div 2^6 = 2^{14}$ buckets. Therefore, the complete index file requires 2^{14} buckets x 833 bytes/bucket = 13,647,872 bytes of storage. The track capacity formula in IBM [110] for the IBM 2314 disk unit indicates that about 7.5 buckets can fit into one track of an IBM 2316 Disk Pack. Therefore, the complete file requires $2^{14} \div 7.5 = 2184$ tracks or 54.6% of an IBM 2316 Disk Pack. If a blocking factor of two is employed, then the corresponding figures are 8 buckets per track, 2046 tracks, and 51.2%. If the buckets are blocked into one-track physical records, then the corresponding figures are 8.76 buckets per track, 1871 tracks, and 46.8%.

6.9.6 Expected Time to Access a Bucket

If an idle IBM 2314 disk unit and an idle channel are assumed then the expected access time for unblocked buckets is 60 ms + 12.5 ms + 3.3 ms = 75.8 ms. The corresponding expected access times for the two cases of blocked buckets discussed above are 78.8 ms and 97.5 ms.

6.10 The Author Hash Index File

6.10.1 Estimation of the Number of Author Index Extries Required for One Million Titles

For the purposes of this discussion, the term "author" is taken to include not only authors in the strict sense, but

also editors, compilers, composers, translators, committee chairmen, and so forth. In addition, an "author name" may be either a personal name or a corporate name.

In the case of personal names, only surnames are entered into the author index. If a query specifies that a match must be found for the initials as well as the surname of a personal author name, then the search program uses the supplied surname, and any other search keys given in the query, to retrieve a set of catalog entries which are then scanned to find those entries that contain both the surname and initials specified in the query. Compound surnames, such as "Von Neumann" and "Stanley-Jones", can be entered as a unit into the author index; or they can be treated in the same manner as corporate names. A complete personal author name, that is, a surname plus initials, is not entered into the author hash index file for two reasons:

i) There are many more different complete author names than there are different author surnames. Hence, the index file and its associated dictionary would be much larger if complete author names were entered in addition to, or in place of, author surnames.

ii) At least one experimental study [129] has demonstrated that library patrons who conduct "known-item" searches usually do not know the initials of the author's name. Consequently, it is felt that complete author names should not be stored in the author name hash index since they would contribute little to the

effectiveness of searches conducted against that index.

In the case of corporate names, each word in the corporate name has a separate entry in the author index and a separate entry in the bits strings file. For example, each word of "International Business Machines Corporation" has an entry in the author index and in the bit strings file. Thus, a user may specify, in the author field of his query, any combination of these four words taken one, two, three, or four at a time. The search program first searches the author index and the bit string file to obtain the entries that correspond to each word in the supplied corporate author name. It then AND's the bit strings to obtain a list of all the catalog entries that contain the specified combination of words in their author fields. Of course, the search program need not limit the searcher to the use of AND logic within the author field of his query. The use of OR and NOT logic, as well as the relations of ADJACENCY and PRECEDENCE, within the author fields of queries would be likely to improve the recall and relevance ratios of most queries that involve corporate author names.

The author index and the bit string file also include entries for common acronyms for corporate names. Thus, "IBM", "SDC", "NASA", and "ASLIB" have entries in the author index and in the bit strings file.

Since it appears that reliable statistics for author names do not exist at present, the following assumptions are

made in order to estimate the number of entries needed in the author index:

i) It is assumed that 2.00×10^5 different surnames are found in the personal author fields of a catalog for a collection of 1.0×10^6 documents.

ii) It is assumed that 4.0×10^4 different words are found in the corporate author fields of a catalog for a collection of 1.0×10^6 documents.

iii) It is assumed that 1.6×10^4 different acronyms are found in the author fields of a catalog for a collection of 1.0×10^6 documents.

Therefore, it is assumed that 2.56×10^5 entries are needed in the author index.

6.10.2 Determination of the Number of Bits Needed in The Major, Minor, and Complete Virtual Hash Address

If $N = 2.56 \times 10^5$ and $\alpha = 1$, then Equation (6.2.6.4) yields $m = \lceil \log_2 (2.56 \times 10^5) = 18$ bits. If $f_c = 2^{-16}$, then Equation (6.2.6.6) yields $v = \lceil [\log_2 (2.56 \times 10^5) - \log_2 2^{-16} - 1] = 33$ bits. It follows from Equation (6.2.6.8) that $p = 33-17 = 16$ bits.

The expected number of collisions at virtual addresses is given by Equation (6.2.5.1) as $\beta = N_c \simeq 4$. Therefore, according to Equation (6.8.2.1) it follows that Prob $[0 \leq C \leq 13] = 0.9999$.

6.10.3 Structure of the Author Hash Index File

As is explained in Section 6.8.3, it is sufficient to describe the structure of the content entries in any hash index file to define that file's structure.

Each content entry in the author hash index file is composed of the following data elements:

i) A two-byte virtual address minor field of which the leading bit is the virtual address collision bit.

ii) A one-byte chain pointer field of which the leading bit is the overflow bit.

iii) A three-byte pointer to an entry in the author word dictionary.

iv) A four-byte pointer to an entry in the author word compressed bit strings file. This pointer is replaced by the entry from the bit strings file if that entry is only four bytes long.

Therefore, each content entry requires ten bytes of storage space.

6.10.4 Number of Content Entries Needed In Each Bucket

If 2.56×10^5 keys are hashed into 2^{18} locations then $\alpha \simeq 1.0$. As was determined earlier in Section 6.2.3, if $b = 64$ and $\alpha = 1.0$ then the required number of content entries is $c = 85$.

6.10.5 Bucket Size

In the preceding section, it was found that each bucket requires 85 content entries. Hence, each bucket is composed of the following data elements:

i) 64 one-byte index entries as is shown in Figure 6.3.2.

ii) 85 ten-byte content entries.

iii) Four one-byte counters which are described in Figure 6.3.2.

Therefore, each bucket requires 918 bytes of storage.

6.10.6 Size of the Author Hash Index File

Since the complete author index file must contain 2^{18} index entries as was determined in Section 6.10.2, and since each bucket contains 2^{6} index entries, the file must contain $2^{18} \div 2^{6} = 2^{12}$ buckets. Therefore, the complete author index file requires $2^{12} \times 918 = 3,760,128$ bytes of storage. The track capacity table in IBM [110] indicates that seven of these buckets fit into one track of an IBM 2316 Disk Pack. Therefore, the complete file requires $2^{12} \div 7 = 585.2$ tracks or 14.6% of an IBM 2316 Disk Pack. If the buckets are blocked into one-track physical records, then the corresponding figures are 7.94 buckets per track, 515.8 tracks, and 12.9%.

6.10.7 Expected Time to Access a Bucket

If an idle IBM 2314 disk and an idle channel are assumed then the expected access time for unblocked buckets is 60ms + 12.5ms + 3.57ms = 76.1 ms. The corresponding expected access time when the buckets are blocked into

one-track physical records is 97.5ms.

6.11 The Coding Scheme for Author Words and Title Words

6.11.1 Introduction

The coding scheme described in this treatment was proposed by H. S. Heaps and L. H. Thiel [113] as an improved version of their earlier coding scheme [112]. The basic principle of this scheme is the same as that used to construct Shannon-Fano codes [130]: the shortest codes are assigned to those words that have the highest frequency of occurrence and the longest codes are assigned to those words that have the lowest frequency of occurrence. If this principle is followed, then the expected length of the resulting codes is less than it would be if all words were mapped into codes of the same length.

6.11.2 Measures of the Efficiency of a Coding Technique

One measure of the efficiency of a coding technique is the expected code length, $\bar{c}$, which is defined by

$$\bar{c} = \sum_{i=1}^{N} f_i \, c_i \qquad (6.11.2.1)$$

where f_i is the relative frequency of occurrence in the material to be coded of the ith word in the vocabulary, c_i is the length of the code assigned to the ith word, and N is the total number of words in the vocabulary. An equivalent

definition of $\bar{c}$, which is often more convenient for computational purposes than Equation (6.11.2.1), is

$$\bar{c} = \sum_{j=1}^{M} p_j c_j \qquad (6.11.2.2)$$

where c_j is the jth code length, $c_j > c_{j-1}$, p_j is the relative frequency of occurrence of codes of length c_j, and M is the number of different code lengths. In terms of f_i's, p_j is defined by

$$p_j = \sum_{i=1}^{N_j} f_i - \sum_{i=1}^{N_{j-1}} f_i \qquad (6.11.2.3)$$

where the N words of the vocabulary are ranked in order of decreasing frequency of occurrence so that $f_i \leq f_{i-1}$ and where N_j is the total number of words mapped into codes of lengths c_1 through c_j.

The expected code length is one measure of the efficiency of a coding technique. A second measure is the ratio, r_s, of the space required to store the data base if the data base is uncoded to the space required if the data base is compressed by means of the coding technique. The ratio r_s, called the data compression ratio, is defined by

$$r_s = \bar{w}/\bar{c} \qquad (6.11.2.4)$$

where $\bar{w} = \sum_{i=1}^{N} f_i w_i$ is the expected word length and w_i is the length of the ith word in the vocabulary.

A third measure of the efficiency of a coding technique is the expected time, $\bar{t}_d$, to decode one code. $\bar{t}_d$

is defined by

$$\bar{t}_d = \bar{t}_a \bar{a} \qquad (6.11.2.5)$$

where $\bar{t}_a$ is the expected time to read a record from the dictionary file and $\bar{a}$ is the expected number of file accesses needed to decode one code.

$\bar{a}$ is defined by

$$\bar{a} = \sum_{i=1}^{N} f_i a_i \qquad (6.11.2.6)$$

where a_i is the number of file accesses needed to decode the code for the ith word in the vocabulary.

These measures of efficiency are used in succeeding sections to evaluate the proposed coding scheme for author words and title words.

6.11.3 Description of the Coding Scheme

The coding scheme proposed by H. S. Heaps and L. H. Thiel [113] and adopted for the purposes of this study is as follows:

i) The most frequent 127 words are coded in one byte as 1XXXXXXX, Xε{0,1}, with 10000000 reserved for use as an escape code. According to a word-frequency distribution in H. Kucera and W. N. Francis [131], this covers 49.572% of all word occurrences in written English text. These one-byte codes are denoted by (j,i). If the bits of the code are numbered from right to left, then bits one through

four form the i part of the code and bits five through seven form the j part of the code. The eighth bit is a flag.

ii) The next most frequent 16,384 words are coded in two bytes as 0XXXXXXX1XXXXXXX, Xε{0,1}. This covers another 45.544% of words occurrences in written English text. These two-byte codes are denoted by (J,I). If the bits of the code are numbered from right to left, then bits one through seven form the I part of the code and bits nine through fifteen form the J part of the code. The eighth and sixteenth bits are flags.

iii) The next most frequent 2,097,152 words are coded in three bytes as 0XXXXXXX0XXXXXXX1XXXXXXX, Xε{0,1}. This covers all but an infinitesimal proportion of the remaining 4.884% of word occurrences in written English text. These three-byte codes are denoted by (K,J,I). If the bits of the code are numbered from right to left, then bits one through fifteen form the J part of the code, and bits seventeen through twenty-three form the K part of the code. The eighth, sixteenth, and twenty-fourth bits are flags.

iv) The 1-bit in the eighth bit position of each code indicates that the next byte on the right is the first byte of a new code.

If $p_1 = 0.49572$, $p_2 = 0.45544$, $p_3 = 0.04884$, $c_1 = 1$, $c_2 = 2$, $c_3 = 3$, and M = 3 are substituted into Equation

(6.11.2.2), then the expected code length, $\bar{c}$, for this coding scheme is found to be 1.55312 bytes/code which is considerably better than the value of exactly two bytes/code achieved by the earlier method reported by H. S. Heaps and L. H. Thiel [112].

The variable-length coding method described above employs a "base" of eight bits. A logical extension is to use a base of four bits or a base of two bits. Calculations based on the word-frequency distribution in H. Kucera and W. N. Francis [131] indicate that a code employing a four-bit base would achieve an expected code length of 1.434 bytes/code. However, any savings gained by using this base or a two-bit base are likely to be offset by increased decoding costs.

6.12 The Dictionary Files

6.12.1 Introduction

Since the structure of the dictionary files for author words and title words is essentially the same as that proposed by H. S. Heaps and L. H. Thiel [112, 113] it is not described in detail in this treatment. However, it is of interest to know the structures of the pointers to the dictionary files, the expected sizes of these files, and the expected access times. The dictionary file for LC call numbers is described in Section 7.2.2.

6.12.2 Structures of the Pointers to the Dictionary Files

As described by Heaps and Thiel [112, 113], each code is a pointer to its corresponding entry in a dictionary file. The K part of each code points to a directory, the J or j part of each code points to an entry in that directory, the directory entry points to a string of uncoded terms, and the I or i part of the code points to a term within the term string. This is shown in Figure 6.12.2.1 for the case of the three-byte (K, J, I) codes. In the case of the one-byte (j,i) and the two-byte (J,I) codes, the K part is omitted from the codes because there is one and only one directory for each of these code sets.

Pointers to the dictionaries from the hash index files are all three bytes in length as mentioned previously in Sections 6.8.4 and 6.10.4. The sixteenth and twenty-fourth bits of each pointer are code flag bits. If these two code flag bits are '00' then the rightmost byte of the pointer is a one-byte (j,i) code and the remaining two bytes contain all zeros. If these flag bits are '01' then the two rightmost bytes are a two-byte (J,I) code that has a corresponding entry in the Compressed Bit Strings File, and the remaining byte contains all zeros. If these flag bits are '10' then the entire three-byte pointer is a three-byte (K, J, I) code. If these flag bits are '11' then the pointer contains a (J,I) code, but the corresponding term has no entry in the Compressed Bit Strings File. Terms corresponding to (j,i) codes never

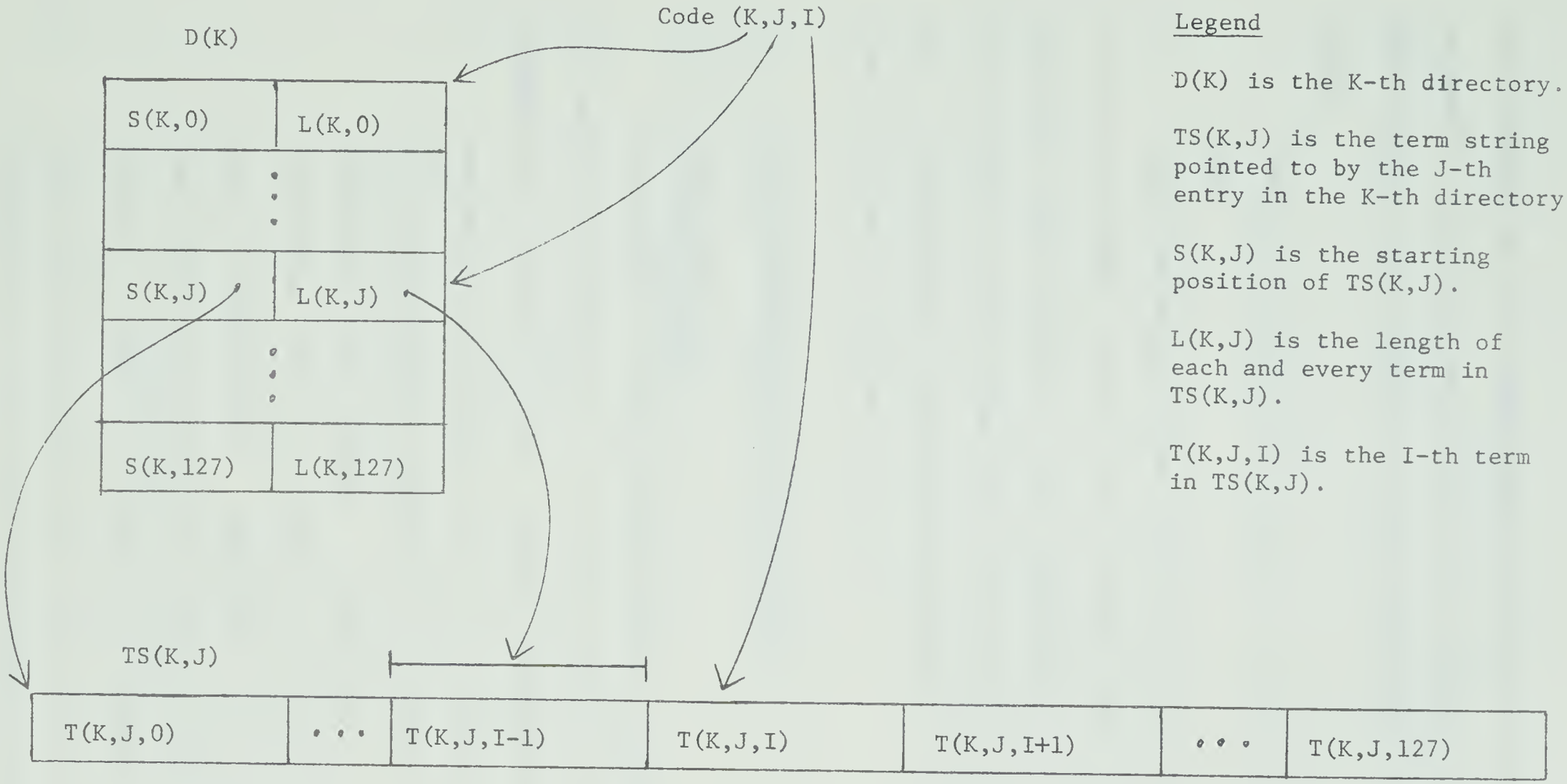


Figure 6.12.2.1 The Relationship of the (K,J,I) Codes to the Structure of the Dictionary Files

have entries in the Compressed Bit Strings File. The frequently occurring words that are assigned (j,i) and (J,I) codes do not have entries in the Compressed Bit Strings File because

i) these entries would require a great deal of space and

ii) high-frequency words are poor search keys.

All (K,J,I) codes, and all (J,I) codes that have '01' as flag bits, have entries in the Compressed Bit Strings File. The structure of the three-byte dictionary pointer field that is contained in the content entries of the hash index files is shown in Figure 6.12.2.2.

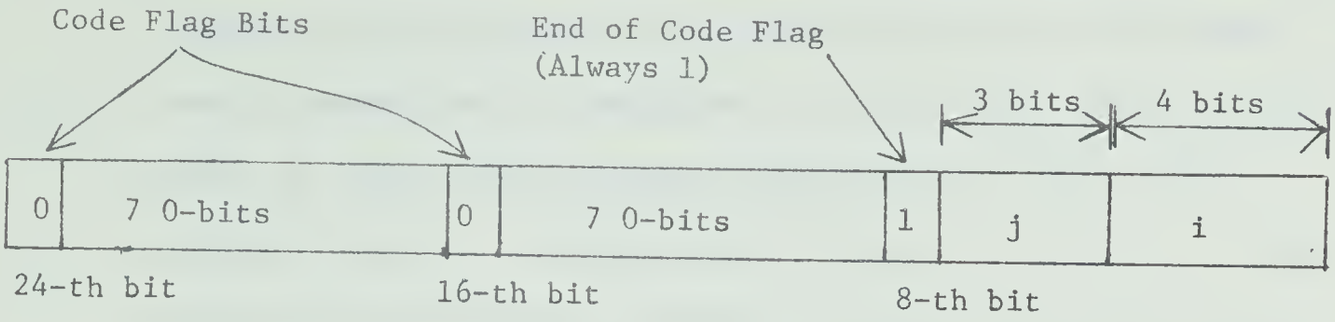
6.13 The Dictionary for Title Words

6.13.1 Estimated Size

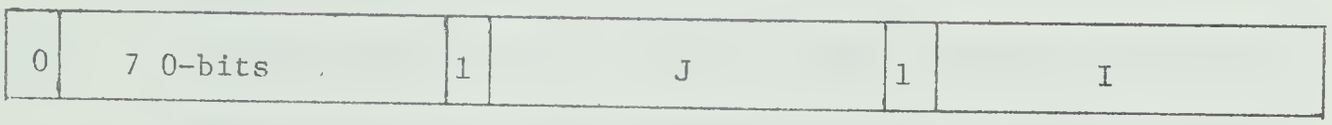
The amount of disk storage space required for the title words dictionary of an on-line catalog system that encompasses 1.0×10^6 titles can be estimated as follows:

i) According to Section (6.8.1), a collection of 1.0×10^6 titles can be expected to encompass 1.8×10^5 different title words. In addition, W. D. Reid [128] found that the mean length of different title words was 7.6 characters. Therefore, the term strings of the title word dictionary can be expected to require $7.6 \times 1.8 \times 10^5 = 13.68 \times 10^5$ bytes of storage.

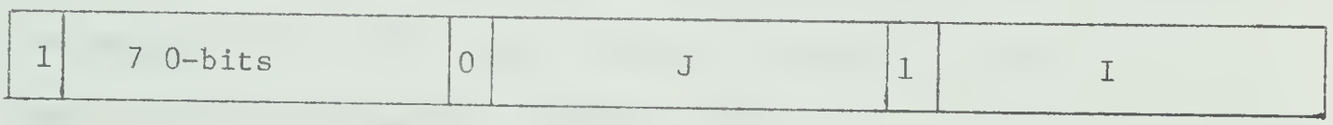
ii) Since each directory for (J,I) and (K,J,I) codes can point to a set of term strings containing 2^{14}



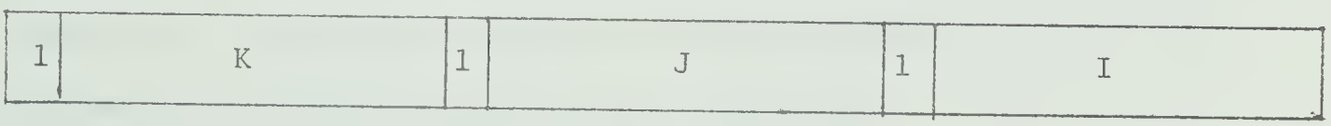
Case One: The (j,i) Codes



Case Two: The Low-Frequency (J,I) Codes



Case Three: The High-Frequency (J,I) Codes



Case Four: The (K,J,I) Codes

Figure 6.12.2.2 The Structure of the Three- Byte Dictionary Pointer Field

words, a dictionary file containing 1.8×10^5 words can be expected to require 11 or 12 such directories. Each of these directories requires 512 bytes of storage space. Consequently, about $12 \times 512 = 6144$ bytes of storage are needed for the (J,I) and (K,J,I) code directories. The (j,i) code directory requires an insignificant amount of space and so the estimated storage required for all directories is taken as 6144 bytes.

iii) If the results of i) and ii) are combined, then the expected size of the complete title word dictionary file is $13.68 \times 10^5 + 0.61 \times 10^5 = 14.29 \times 10^5$ bytes.

If the logical records of the title word dictionary are mapped into 1000-byte physical records, then the file requires 1429 of these physical records. According to the track capacity table in IBM [110], about six 1000-byte physical records fit into one track of an IBM 2316 Disk Pack. Therefore, the file requires in the neighborhood of 238 tracks or 5.95% of an IBM 2316 Disk Pack.

6.13.2 Expected Decoding Time

According to Equation (6.11.2.5), the expected decoding time, $\bar{t}_d$, is the product of the expected time, $\bar{t}_a$, to read one record from the dictionary file and the expected number of file accesses, $\bar{a}$. In this discussion, CPU time is considered to be insignificant in comparison with the time needed for file accesses.

The expected time to access one 1000-byte physical record in the dictionary file is $\bar{t}_a$ = 60ms + 12.5ms + 4.1ms = 76.6ms if an idle IBM 2314 disk unit and an idle channel are assumed.

The expected number of file accesses required per code during the decoding of title fields in compressed catalog entries depends upon the following:

i) The manner in which the dictionary is partitioned between core memory and disk memory. If part of the dictionary is stored in core memory then it need not be read into core memory every time a code references it.

ii) The distribution of word frequencies in the data base.

iii) The manner in which the directories and term strings are mapped into physical records on the disk.

iv) The number of catalog entries retrieved in response to a query. If more than one catalog entry is retrieved, then the titles of the "hits" can be expected to have some words in common. Such common words need be decoded only once for the set of hits.

The following assumptions are made to facilitate the calculation of $\bar{a}$:

i) The word-frequency distribution in H. Kucera and W. N. Francis [131] applies to title words.

ii) The (j,i) code directory and term string are stored in a single physical record so that they may be read

with a single disk access.

iii) Each directory for (J,I) or (K,J,I) codes can be obtained with one disk access.

iv) Required sections of term strings for (J,I) or (K,J,I) codes can be obtained with one disk access.

v) Only one catalog entry is retrieved in response to a query. Therefore, the estimated values of $\bar{a}$ are upper bounds for $\bar{a}$ because codes common to two or more hits need be decoded only once.

Table 6.13.2.1 shows the results of the application of Equations (6.11.2.5) and (6.11.2.6) to the following six different core-disk partitions of the dictionary file:

Case One: No part of the file is core-resident.

Case Two: The directory and the term strings for (j,i) codes are core-resident.

Case Three: The directory and the term strings for (j,i) codes and the directory for (J,I) codes are core-resident.

Case Four: All directories and the term strings for (j,i) codes are core-resident.

Case Five: The directory and term strings for (j,i) codes, the directory for (J,I) codes, and the dictionary entries for the 128 most frequently used (J,I) codes are core-resident.

Case Six: All directories, the term strings for (j,i) codes, and the dictionary entries for the 128 most frequently used (J,I) codes are core-resident.

TABLE 6.13.2.1

Comparison of Six Different Core-Disk Partitions of the Title Word Dictionary

Case[1]	(j, i) codes				
	Accesses			Probability of Codes	Expected Number of Accesses
	Directory	Term	Strings		
One	1	0		0.49572	0.49572
Two	0	0		0.49572	0
Three	0	0		0.49572	0
Four	0	0		0.49572	0
Five	0	0		0.49572	0
Six	0	0		0.49572	0

[1]Each case is described in Section 6.13.2

Continued

TABLE 6.13.2.1 (continued)

Case[1]	(J, I) codes			
	Accesses		Probability of Codes	Expected Number of Accesses
	Directory	Term Strings		
One	1	1	0.45544	0.91088
Two	1	1	0.45544	0.91088
Three	0	1	0.45544	0.45544
Four	0	1	0.45544	0.45544
Five	0	0	0.06147	0
		1	0.39397	0.39397
Six	0	0	0.06147	0
		1	0.39397	0.39397

[1]Each case is described in Section 6.13.2

Continued

TABLE 6.13.2.1 (Continued)

Case[1]	(K, J, I) codes			
	Accesses		Probability of Codes	Expected Number of Accesses
	Directory	Term Strings		
One	1	1	0.04884	0.09768
Two	1	1	0.04884	0.09768
Three	1	1	0.04884	0.09768
Four	0	1	0.04884	0.04884
Five	1	1	0.04884	0.09768
Six	0	1	0.04884	0.04884

[1]Each case is described in Section 6.13.2

Continued

TABLE 6.13.2.1 (Continued)

Case[1]	All Codes			
	Expected Number of Accesses	Expected Decoding Time (ms)	Expected Time to Decode an Average Title That Contains 5.5 Codes (ms)	Dedicated Core Memory (bytes)
One	1.50428	115.20	633.6	0
Two	1.00856	77.26	424.9	989
Three	0.55312	42.37	233.0	1501
Four	0.50428	38.63	212.5	7133
Five	0.49165	37.66	207.1	2474
Six	0.44281	33.92	186.6	8106

[1]Each case is described in Section 6.13.2

The last column of Table (6.13.2.1) gives an estimate of the amount of core memory that must be dedicated to implement each of the six cases.

The results tabulated in Table (6.13.2.1) lead to the conclusion that Case Three is probably the best core-disk partition of the dictionary file.

6.13.3 The Expected Compression of the Title Fields of Compressed Catalog and the Expected Saving in File Storage Costs

According to Section 6.11.3 the expected code length for title words is $\bar{c}$ = 1.55312 bytes/code and according to W. D. Reid [128] the expected length of title words is $\bar{w}$ = 5.6 bytes/word. Therefore, according to Equation (6.11.2.4), the expected data compression ratio for title words is r_s = 5.6/ 1.55312 = 3.67. This value of r_s implies that the compressed title fields occupy only 27.2% as much space as is occupied by the uncompressed title fields.

This saving of space may be translated into a saving of money as follows. According to W. D. Reid [128] the average number of words per title is 5.5. Therefore, the total space occupied by one million uncompressed title fields is 1.00×10^6 titles x 5.5 words/title x 5.6 bytes/ word = 3.08×10^7 bytes. On the other hand, if these title fields are compressed by means of the method described in Section 6.9.3, then they occupy 1.00×10^6 titles x 5.5 codes/title x 1.55312 bytes/code = 8.54×10^6 bytes. The current rate for

disk storage at the University of Alberta Computing Center is \$0.327 page/month. Therefore, the corresponding disk storage costs for these two cases are 3.08×10^7 bytes x $\frac{0.327}{4096}$ $\frac{\text{dollars}}{\text{byte-month}}$ $\simeq$ \$2459 per month and 8.54×10^6 bytes x $\frac{0.327}{\text{byte-month}}$ $\simeq$ \$682 per month. Consequently, the saving in disk storage cost is \$1777 per month. Of course, this saving in file storage cost is partly offset by the costs of encoding and decoding of the title fields of the compressed catalog entries.

6.14 The Dictionary for Author Words

6.14.1 Estimated Sizes

The amount of disk storage space required for the author words dictionary of an on-line catalog system that encompasses 1.0×10^6 titles can be estimated as follows:

i) According to Section 6.10.1, a collection of 1.0×10^6 documents can be assumed to include 2.00×10^5 different surnames, 4.0×10^4 different words in corporate author names, and 1.6×10^4 different acronyms. Furthermore, since reliable information about author names is scarce, it is assumed that a) the average length of an author surname is 7.0 characters, b) the average length of a word in corporate names is 6.0 characters, and c) the average length of an acronym is 4.0 characters. Therefore, the term strings of the author word dictionary can

be expected to occupy $2.00 \times 10^5 \times 7.0 + 4.0 \times 10^4 \times 6.0 + 1.6 \times 10^4 \times 4.0 = 1.704 \times 10^6$ bytes.

ii) Since each directory for (J,I) and (K,J,I) codes can point to a set of term strings containing 2^{14} words, a dictionary file containing 2.56×10^5 words can be expected to require about 16 such directories. Each of these directories requires 512 bytes of storage space. Consequently, about $16 \times 512 = 8{,}192$ bytes are needed for the storage of the (K,J,I) and (J,I) code directories. The (j,i) code directory requires an insignificant amount of space and so the estimated storage required for all directories is taken to be 8,192 bytes.

iii) If the results of i) and ii) are combined, then the expected size of the complete author word dictionary file is $1.704 \times 10^6 + 0.0082 \times 10^6 = 1.712 \times 10^6$ bytes.

If the logical records of the author word dictionary are mapped into 1000-byte physical records, then the file requires 1712 of these physical records. According to the track capacity table in IBM [110], about six 1000-byte physical records fit into one track of an IBM 2316 Disk Pack. Therefore, the author word dictionary requires in the neighborhood of 286 tracks or 7.15% of an IBM 2316 Disk Pack.

6.14.2 Expected Decoding Time for Author Words

According to Equation (6.11.2.5), the expected decoding time, $\bar{t}_d$, is the product of the expected time, $\bar{t}_a$, to read one record from the dictionary file and the expected number of file accesses, $\bar{a}$. In this discussion, CPU time is considered to be insignificant in comparison with the time needed for file accesses. In Section 6.13.2, $\bar{t}_a$ was found to be 76.6ms. In addition, the factors that affect $\bar{a}$ were discussed in that section. Since, the frequency distribution for author words is not known, the exact value of $\bar{a}$ cannot be determined. However the upper bound on $\bar{a}$ is 2.00 accesses/code and, therefore, the upper bound on $\bar{t}_d$ may be assumed to be 153.2 ms per code.

6.14.3 The Expected Compression of the Author Fields of the Compressed Catalog

If it is assumed that $\bar{w}$ = 5.0 bytes/word and $\bar{c}$ = 3.0 bytes/code, then according to Equation (6.11.2.4) the expected data compression ratio for author words is r_s = 2.0. This value of r_s implies that the compressed author fields of the compressed catalog require only 50% as much space as is required by the uncompressed author fields.

6.15 The Compressed Catalog File

The entries of the Compressed Catalog File contain information equivalent to that contained in standard card catalog entries, except that certain information which is

likely to change is kept in the files of the circulation system. This volatile information includes the number of copies held by the library, the accession numbers of those copies, and the locations of the various copies. The structure of the entries in the Compressed Catalog will not be described at this point, but the overall structure will be similar to that of the MARC II format [111] and the ILR format [78], except that the information contained in most fields will be coded to conserve space.

The logical operations of the catalog query system are designed to produce a series of item numbers which represent those entries in the Compressed Catalog that satisfy the conditions of a given query. Hence, it is necessary to provide some means of translating the item numbers into the addresses of the required variable-length records in the Compressed Catalog. There are several methods of arranging the records of the Compressed Catalog File so that this translation may be performed; however, only one method will be discussed in any detail below.

The method outlined here emphasizes speed of access at the expense of a certain wastage of disk space and at the expense of the loss of some members of the set of all possible item numbers. Additionally it cannot be used without some knowledge of the distribution of the lengths of the entries in the Compressed Catalog.

Basically, the method is as follows. First, the total disk storage space allocated for the file is divided into

fixed-size physical blocks which are large enough to contain a dozen or more logical records. Second, the minimum length of the entries to be stored in the file is determined. Third, the number of such minimum length entries that will fit into a single physical block is determined. This number, call it N, is the maximum number of logical records that may be fitted into a single physical record. Fourth, insert the logical records of the file into the physical blocks in such a manner as to minimize the amount of space wasted at the end of each physical block. This means that several physical records are filled in parallel; a note is kept of the number of items currently in each and of the amount of space remaining in each. Fifth, as each entry or item is inserted into a physical record it is given an item number based on the relative number of the physical record into which it has been placed and on its relative position within that physical record.

For example, if an item is inserted into physical block j as item number k within that block, then it is assigned the item number i = j*N+k. This item number i is then used as a pointer to this entry and the bit strings for terms contained in the title and other coded fields of this entry will have the ith bit turned on. Hence, if a query operating on these term bit strings should require the document represented by the ith bit, the address of that document could be determined as follows. Divide the item number, i, by N to obtain the relative number, j, of the physical record that

contains the ith item or entry or document and take the remainder k = i-j*N as the relative number of the required logical record within the jth physical record. Of course, if N should be equal or very nearly equal to a power of two, then these calculations are immensely simplified since register shifting operations can replace the multiplication and division operations.

The method as outlined above can be called the pseudo-direct calculation method since it mimics the direct calculation method which can be used with fixed-length logical records. The direct calculation method is described in IBM [110].

The pseudo-direct calculation method, of course, requires that each entry or item be prefixed by its length. For the expected sizes of compressed catalog entries, one byte would be sufficient to contain the length of any catalog entry.

All of the alternative methods which could be used here require the use of some form of directory or cross-reference file. Thus, when an item number is translated into the address of the corresponding item or entry, an extra disk access is needed to obtain the appropriate section of the directory. The pseudo-direct calculation method avoids this extra disk access required to obtain the directory entry at the expense of wasting a certain amount of space at the ends of physical blocks when that space is insufficient to contain even minimum length catalog entries. However, this wastage is

at least partially offset by the saving of space which would otherwise be used to store a directory.

The second expense mentioned above is the loss of some item, or entry numbers, from the total set of item numbers. Since it is usually not possible to store the maximum number, N, of items or entries in each fixed-size physical block, some item numbers cannot be assigned to entries and hence will be lost.

The total amount of storage space wasted at the ends of physical blocks can be reduced by increasing the size of the physical blocks, but an upper limit to this size is dictated by such parameters as the track size in mass storage devices that do not allow physical records to cross track boundaries, the amount of core storage available for buffers, and the time required to transmit the physical record between disk storage and core storage. An optimum physical record size for the Compressed Catalog cannot be specified until knowledge of the distribution of lengths of the compressed catalog entries is available, but in order to provide a good blocking factor these physical records should be from 1K to 1 track in length.

CHAPTER VII

PROPOSED FILE STRUCTURE FOR THE REAL-TIME CIRCULATION SUBSYSTEM

7.1 Factors that Influence the Design of the File Structure

7.1.1 The Characteristics of the Library for Which the System is Designed

The file structure that is described later in this chapter is designed to support a real-time circulation subsystem that is suitable for use in a large academic library. It is assumed that such a library contains 1.0×10^6 titles, 1.2 volumes per title, and that it serves 5.0×10^4 patrons. It is further assumed that no more than 7% of the total number of volumes in the library's collection are charged out to patrons at any one time. These assumptions are based on statistics gathered by the Systems Planning and Development Department of the Cameron Library of the University of Alberta [132].

7.1.2 The Characteristics of the Circulation Subsystem

Among the characteristics of the Real-Time Circulation Subsystem that influence the design of its supporting file structure are the following:

i) Since the circulation subsystem is part of an integrated system that touches upon all facets of the

library's activities, the files of the circulation subsystem should not duplicate information contained in the files of the On-Line Catalog Subsystem or in the files of the Technical Services Subsystems. Conversely, the files of these other subsystems should not duplicate information contained in the files of the circulation subsystem. Whenever possible files should be shared by all of the subsystems of the overall library automation system. One of the primary principles of the file organization discussed in this chapter is that information which is likely to change is stored in the files of the circulation subsystem and not in the Compressed Catalog File. Such information includes the number of volumes in a journal and the number of copies of a book. This is done so as to minimize the need to alter entries in the Compressed Catalog File.

ii) The file structure that supports the circulation subsystem must encompass monographs, closed and open multi-volume sets, and bound and unbound journals. In addition, it must be readily extendable to include other materials handled by the library. These other materials include phonograph records, maps, reports, microfiche, and microfilm.

iii) The circulation subsystem must be able to automatically prevent the borrowing of books by delinquent borrowers, and it must also automatically prevent

the use of stolen, lost, outdated, or invalid patron identification badges.

iv) The circulation subsystem must be able to provide the functions of all of the on-line commands discussed in Sections 3.6 and 3.7 except for the commands SEARCH, RESERVE, AVAILABLE, and DEMAND. A much advanced version of SEARCH would be provided by the On-Line Catalog Subsystem, RESERVE would be dropped entirely, and the functions of AVAILABLE and DEMAND would be transferred to batch-mode programs. Furthermore, the circulation subsystem should provide four additional on-line commands: CANCEL, RECALL, CHANGE, DISPLAY, and SET. CANCEL would allow the librarian or the patron to cancel a request, or reservation, for a book; RECALL, would allow the librarian to specify that a particular book be recalled; CHANGE, would allow the librarian to modify the status of a book, the lending class of a book, or the holding library of a book; DISPLAY, would provide the librarian with all of the information associated with a particular book; Set would allow the librarian to change various system parameters, such as the number of days that are to elapse between the time a book becomes overdue and the time an overdue notice is generated by the circulation system.

v) Since the formats of the on-line commands necessarily depend upon the type or types of terminals used by

the circulation subsystem, no attempt is made here to rigidly define these formats. However, in the interests of more efficient use of the computer system, most of the commands should require the use of accession numbers to identify books and the use of patron identification numbers to identify patrons. This contrasts sharply with the employment of author names, LC call numbers, and patron names in the commands defined earlier in Chapter III. Those commands that still require the use of LC call numbers rather than accession numbers are REQUEST, FIND, CANCEL, ENQUIRE, and RECALL. Another major change from the formats specified in Chapter III is the elimination of passwords on individual commands. Instead, the commands would be grouped into packages and global passwords would be used to protect packages that contain commands intended for use by the librarian only.

vi) Since most people dislike standing in line for service at library check-out desks, the Real-Time Circulation Subsystem should give top priority to the servicing of the commands LOAN and RENEW.

7.1.3 The Desirable Global Characteristics of the File Structure

The file structure that supports the circulation subsystem should satisfy the following general requirements:

i) It must be economical in its use of disk storage space.

ii) It must allow rapid access to the disk records needed for the handling of on-line transactions.

iii) It should allow for the efficient performance of all on-line and off-line functions.

iv) Its design must not be dependent upon a particular operating system or upon a particular file management system. However, the assumption is made that the system is to be implemented on an IBM 360 series computer and that the files of the circulation system are to be stored on an IBM 2314 disk storage unit.

v) It should be such that the addition of new entries to any file does not involve resorting the file and, as a consequence, the changing of a large number of pointers to the entries in the file. This stipulation makes continuous, on-line, updating of the files both feasible and economical.

vi) It should be such that the entry points used by the most frequent on-line commands transform directly into the disk addresses of the required records. Thus, if the program is given the accession number of a book or the identification number of a patron, it should be able to directly calculate the address of the required accession record or patron record without referring to a drum, or disk, resident index or directory of any kind. This implies that any one of these accession or patron records can be obtained in a single disk access. It also implies that these

records must remain in fixed positions within their respective files.

vii) It should be such that the amount of time needed to access a record in any file is an inverse function of its expected frequency of use.

These criteria, and those outlined in Sections 7.1.1 and 7.1.2, have been applied to the design of the file organization that is described below.

7.2 Description of the Files

7.2.1 Overview

The proposed file structure for the Real-Time Circulation Subsystem encompasses eight files:

i) The LC Call Number File.
ii) The Accession File.
iii) The Reservations File.
iv) The Loans File.
v) The Patron File.
vi) The Master Patron File.
vii) The Accession Directory File.
viii) The LC Call Number Hash Index File.

The interrelationships of these files and the Compressed Catalog File, the principal file of the On-Line Catalog Subsystem, are shown in Figure 7.2.1.1. This figure also shows the entry points used by the on-line commands of the Real-Time Circulation Subsystem. The structures and functions of the first seven of these files are described in the sections

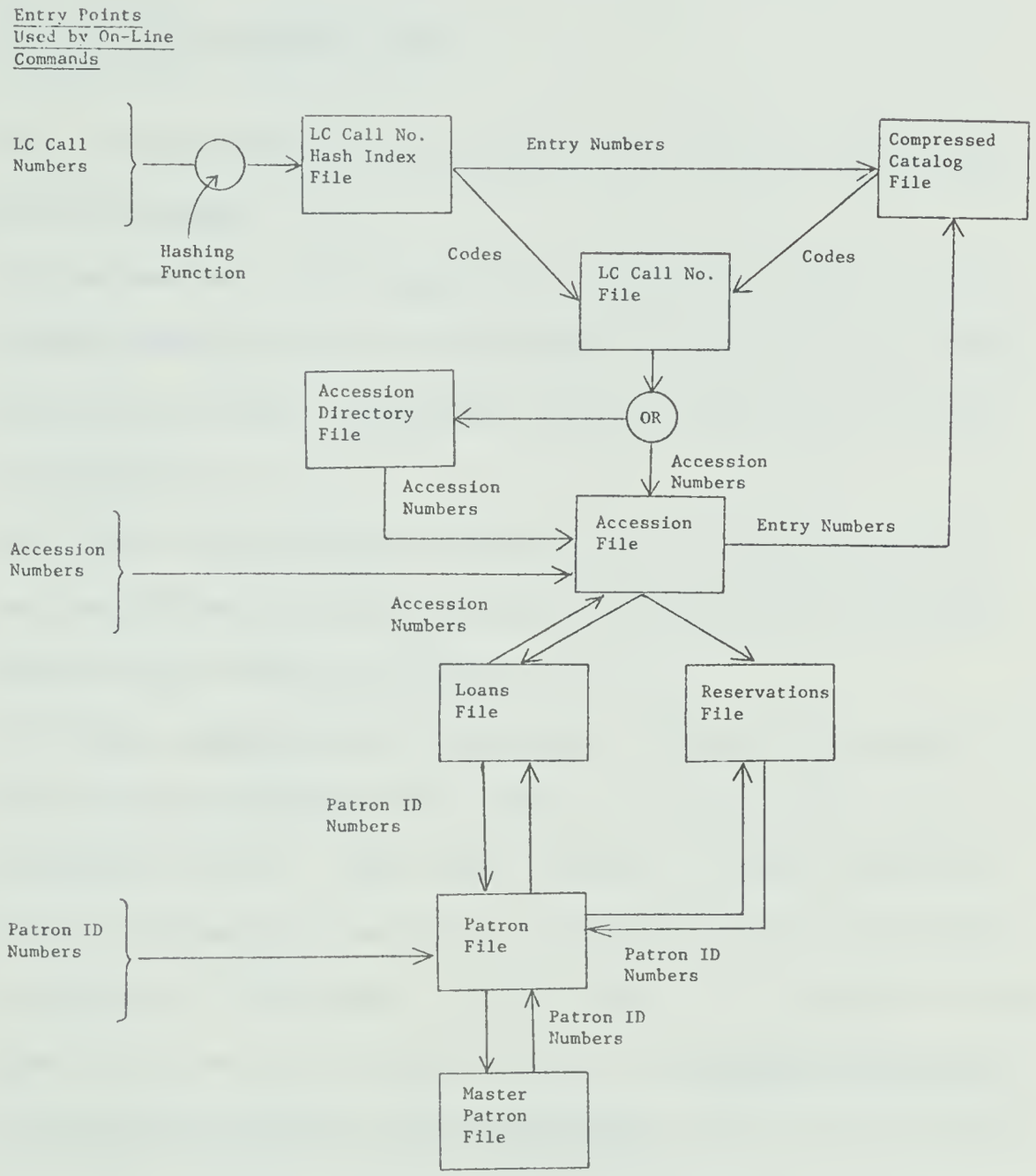


Figure 7.2.1.1 The Files of the Real-Time Circulation Subsystem and Their Relationships with Each Other and the Compressed Catalog File

that follow. The LC Call Number Hash Index File was previously described in Section 6.2.13.

7.2.2 The LC Call Number File

The LC Call Number File has three primary functions which are as follows:

i) It serves as a dictionary for the three-byte LC call number codes that are contained in the LC Hash Index File and in the Compressed Catalog File, as is shown in Figure 7.2.1.1.

ii) It links the files of the On-Line Catalog Subsystem to the files of the Real-Time Circulation Subsystem, as is also shown in Figure 7.2.1.1.

iii) It links together the dispersed accession records for all volumes and all copies of each title that is entered into it. The term "title" is here taken to mean a monograph, an open or closed multi-volume set, a journal, a document, or the like. An example of an open multi-volume volume set is the Annual Review of Information Science and Technology which is published annually, and an example of a closed multi-volume set is The Advanced Theory of Statistics by Kendall and Stuart which was published in three volumes.

The entries in this file, which is also called the LC File, are variable in length and contain the following data elements:

i) The variable-length LC call number which uniquely identifies the title of the monograph, journal, document, or multi-part work that corresponds to this entry. Neither the copy number nor the volume number are included as part of this LC call number.

ii) One byte which gives the length of the LC call number.

iii) One byte which indicates the type of item referred to by the record and, at the same time, the format of the record itself. These item types and corresponding record types are: a) monographs b) closed multi-volume sets, c) open multi-volume sets, and d) bound and unbound journals. This type-byte also contains an indicator that indicates whether the accession record pointer points directly to the accession record or indirectly through an entry in the accession directory, which is described in Section 7.2.8.

Figures 7.2.2.1 through 7.2.2.4 illustrate the structures of the four types of records that are contained in the LC Call Number File.

The records in the LC File are unordered because the addition of new records to an ordered LC File requires re-sorting of the file and the changing of pointers in at least three other files, the LC Hash Index File, the Accession File, and the Compressed Catalog File. Changing the pointers could be avoided, of course, by using a directory. Each pointer to this file from another file would point to a fixed entry in a directory and that directory entry would contain the

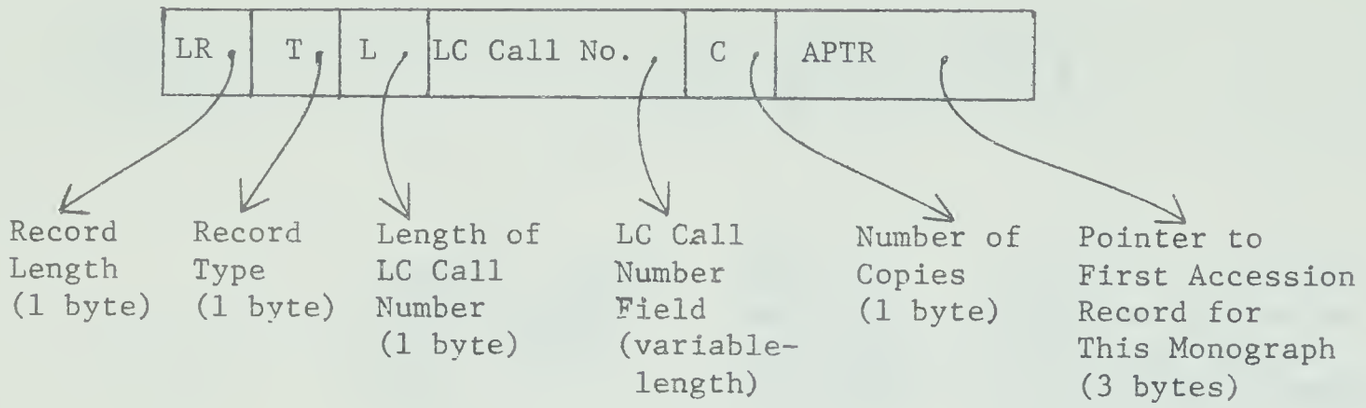


Figure 7.2.2.1 Structure of Records for Monographs in the LC Call Number File

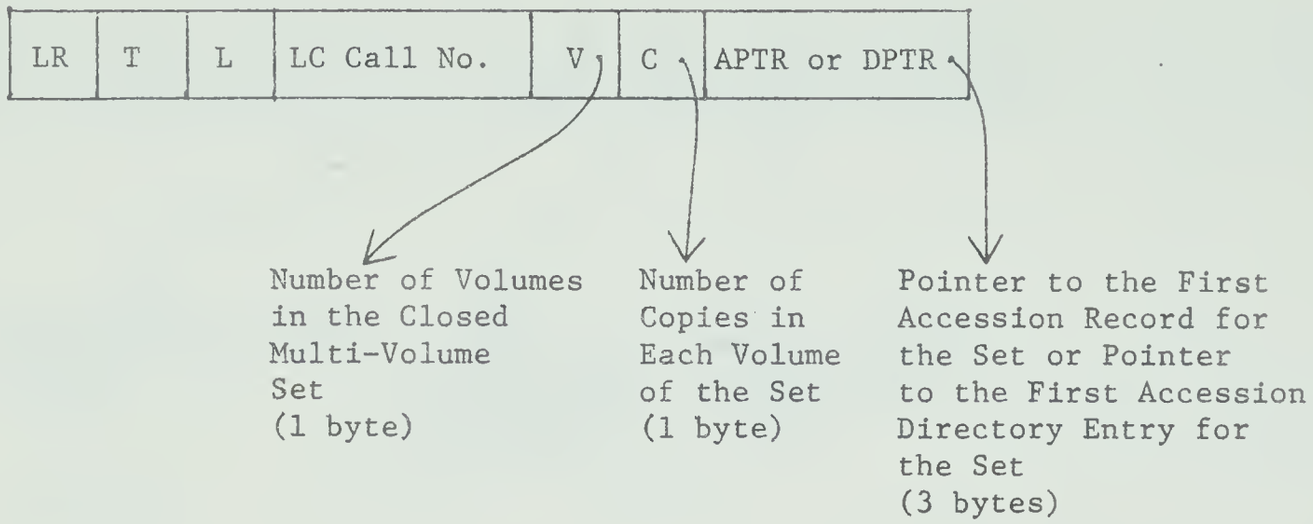


Figure 7.2.2.2 Structure of Records for Closed Multi-Volume Sets in the LC Call Number File

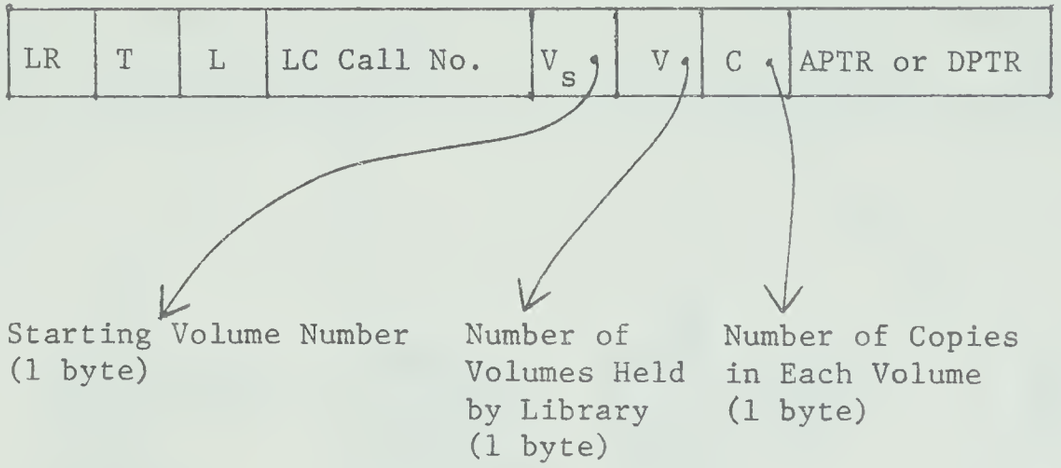


Figure 7.2.2.3 Structure of Records for Open Multi-Volume Sets in the LC Call Number File

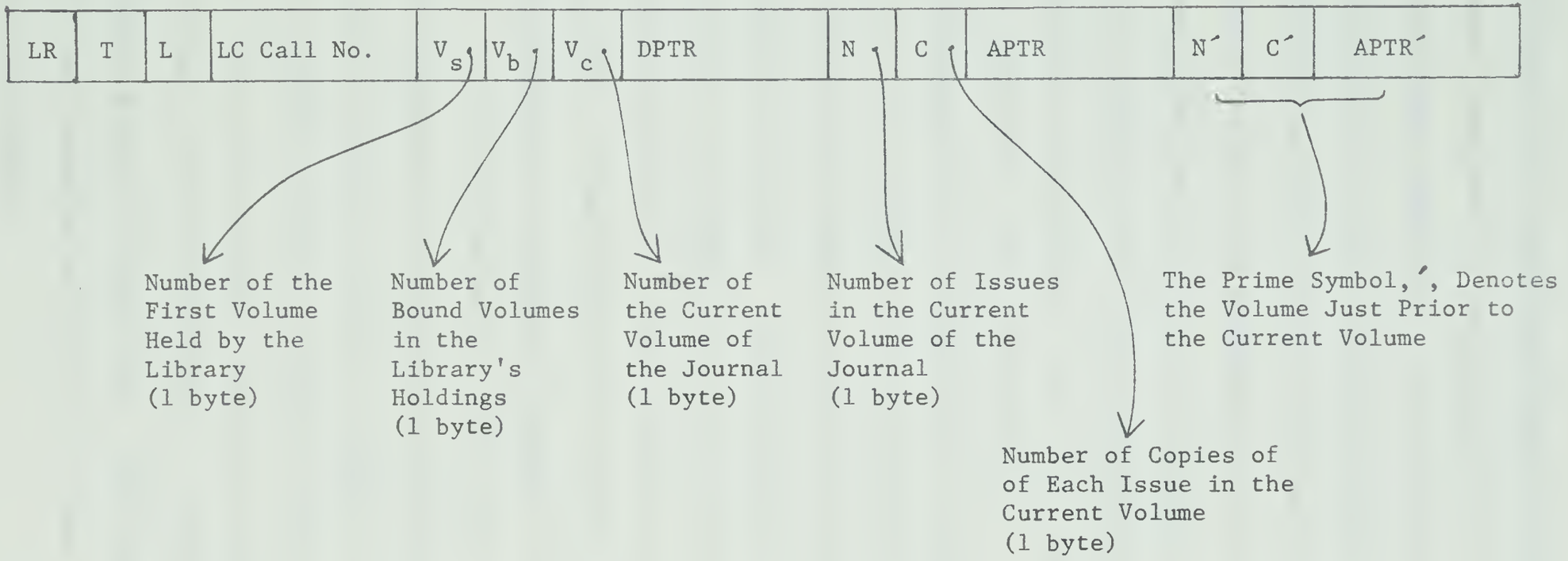


Figure 7.2.2.4 Structure of LC File Records for Both Bound and Unbound Periodicals

actual address of the required entry in the LC File. Aside from the cost involved in resorting the LC File, the directory approach has three penalties. The first is that extra disk space is required to store the directory, the second is that every reference to the LC File requires an extra disk access, and the third is that the addresses in the directory must be changed everytime the LC File is resorted. These penalties outweigh any benefits that may be gained by ordering the LC File. Therefore, new records are added only to the end of the LC File and in no particular order.

The logical LC records shown in Figures 7.2.2.1 through 7.2.2.4 are mapped into the physical disk file as follows. Thirty-two logical records are grouped together to form one physical record. These physical records are, of course, variable in length. They are written on the disk in undefined format and with track overflow turned on so that a record may start on one track and continue onto the next track. According to a table of lengths of LC call numbers given in J. L. Cunningham et al. [78], the minimum length of an LC call number may be taken as six characters. Using this and the information given in Figure 7.2.2.1, the minimum length of a logical record in the LC File may be taken to be 13 bytes. Thus, the minimum length of a physical record in the LC File is (32 x 16) 416 bytes. According to the Track Capacity Table given in IBM [110], less than 14 such minimum-length physical records will fit onto one 7294-byte track of an IBM 2316 Disk Pack.

Each pointer to a logical record in the LC File is three bytes in length and is structured as follows:

i) The first 15 bits indicate the track that contains the logical record. This track number is actually relative to the start of the file and, hence, is not an absolute track address. It is converted to an absolute track address by referring to an entry in a file table which gives the absolute address of the start of the file.

ii) The next four bits indicate the physical record within the track. These four bits are quite sufficient since it is expected that no more than 14 physical records will ever fit onto a track.

iii) The last five bits of the pointer indicate the logical record within the physical record.

This pointer structure is illustrated in Figure 7.2.2.5.

Since the logical records of the LC File are variable in length, the length of each item must be made available in some way to the file management routines. One way is to prefix each logical record by its length as is shown in Figures 7.2.2.1 through 7.2.2.4, and a second way is to include the length of each logical record in all pointers to it. Figure 7.2.7.4 shows one form of this second method. A third method, illustrated in Figure 7.2.2.6, is to place the lengths of all of the logical records together at the beginning of the physical record.

As is shown in Figures 7.2.2.1 through 7.2.2.4, and

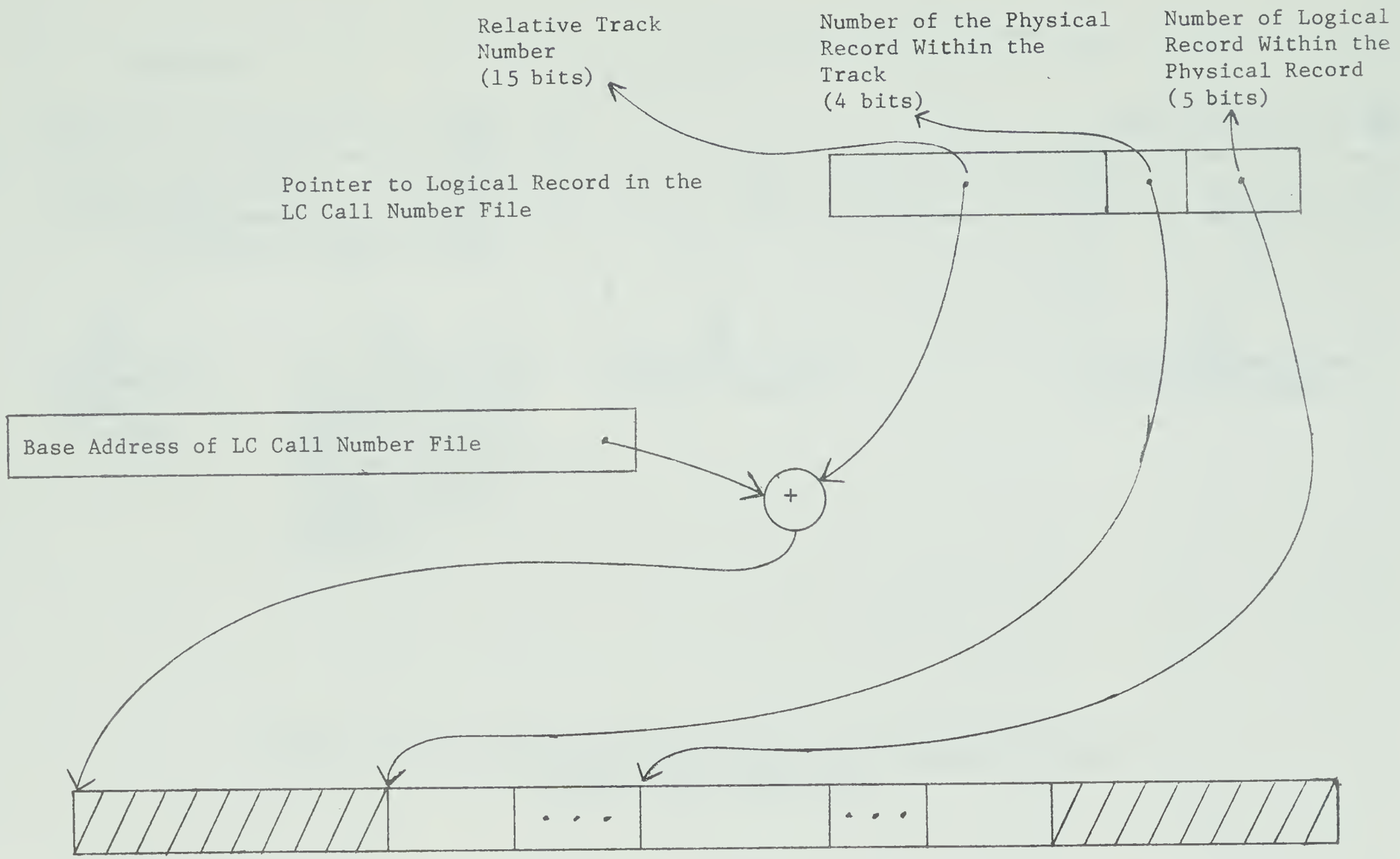


Figure 7.2.2.5 Structure of Pointers to the LC Call Number File

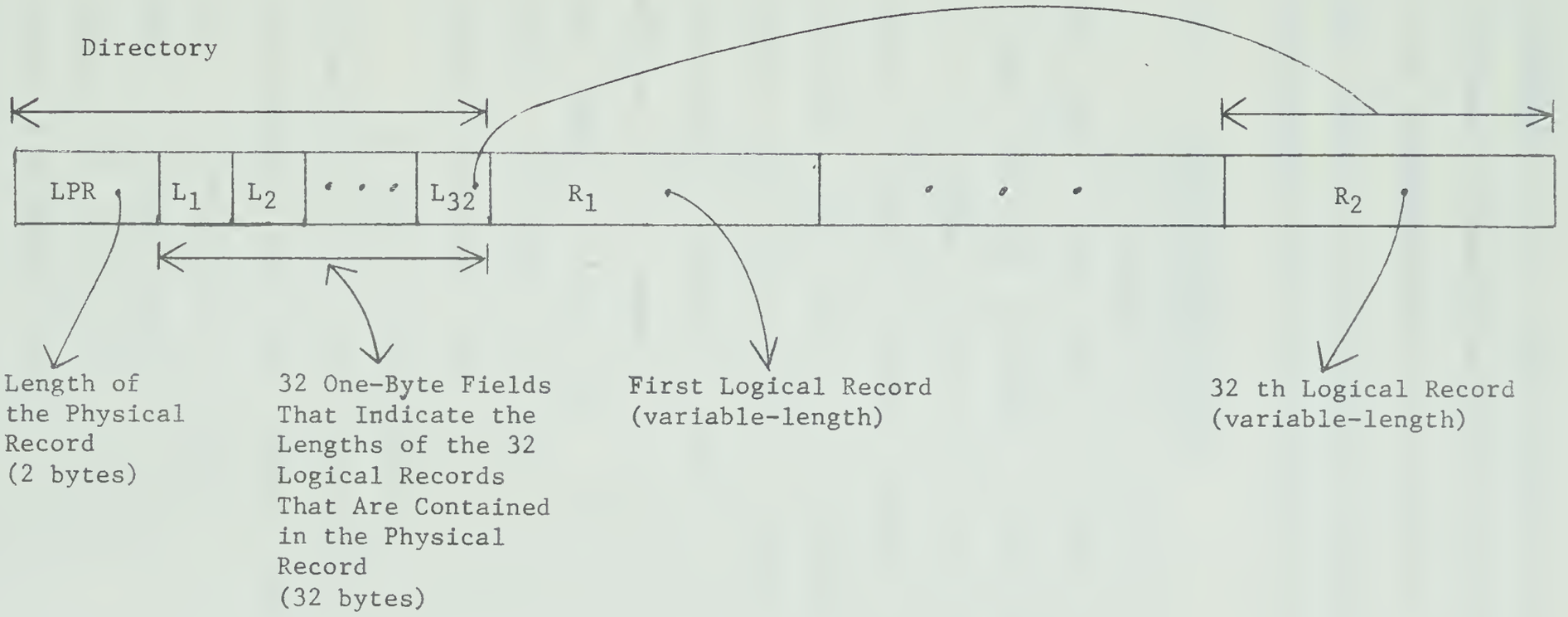


Figure 7.2.2.6 An Alternative Structure for the Physical Records of the LC Call Number File

in Figure 7.2.1.1, each record in the LC Call Number File includes a three-byte field that contains either a pointer to the Accession File or a pointer to the Accession Directory File, which in turn contains pointers to the Accession File. The Accession File is discussed in the next section.

7.2.3 The Accession File

The Accession File contains one 12-byte record for each copy of each volume of each title held by the library. As each new item is added to the library's collection a record for it is added to the end of the Accession File and the number of this new record becomes the accession number of the new item. This one-to-one correspondence between accession numbers and record numbers, coupled with the fixed size of each record, allows the direct calculation of the disk address of the record for a particular item from the accession number of that item. Thus, the on-line commands LOAN, RETURN, and RENEW that require rapid access to these accession records can obtain them with a single disk access.

If two or more copies of a monograph are acquired at the same time then they are assigned consecutive records in the Accession File and, hence, consecutive accession numbers. This is illustrated by Figure 7.2.8.3, which also shows the manner in which entries in the LC File point to such a contiguous set of Accession Records. The allocation of accession records and their linkage to their parent records in the LC Call Number File is explored further in Section 7.2.8.

Each record in the Accession File contains the

following:

i) A three-byte pointer to an entry in the Compressed Catalog File.

ii) A one-byte field that contains status information about the item that corresponds to this accession record. Each of the eight bits in this byte is a status indicator. The first bit is called the On-shelf Bit and indicates, if on, that the technical processing of the item, including ordering, receiving, accessioning, binding, labelling, and cataloging, has been completed and that the item has been placed on the library's shelves and is, therefore, available for use by the library patrons. The second bit is called the Added-Copy Bit and, if on, indicates that the item which corresponds to this accession record is an added copy of a particular volume of a particular title held by the library. What is indicated by the next six bits depends on whether or not the On-Shelf Bit is on. If the On-Shelf Bit is off, then bits three through eight indicate which stages have been completed in the technical processing of the item. If the On-Shelf Bit is on, then bits three through eight, if on, indicate, respectively, that a reservation exists for the item, that a hold for a patron has been placed on the item, that the current loan of the item has been renewed, that a recall notice has been sent, that the item is overdue, and that the item has been

lost, stolen, or retired.

iii) A four-bit field which contains a code that identifies the branch of the library where the item is held.

iv) A four-bit field that defines the borrowing class of the item which may be, among others, reserve reading room, reference work, three-day loan, two-week loan, or unlimited loan.

v) A three-byte reservations pointer which, if non-zero, either points to the head of a reservations chain or else to another accession record which contains the actual pointer to the head of the reservations chain. The reservations chain is contained in the Reservations File which is discussed in the next section. The second case applies if and only if the accession record is for a copy other than copy number one. The leading bit of the pointer is '0' if the first case applies and '1' if the second case applies. As is indicated in Section 7.2.4, a pointer to an entry in the Reservations File requires only two-bytes of storage space. Therefore, if the first case applies then the pointer to the Reservations File is right-justified within the three-byte pointer field.

vi) A one-byte loans counter which indicates the number of times that the item has been borrowed since it was placed on the library shelves.

vii) A three-byte pointer to an entry in the Loans File which will be described presently. If the Hold Bit,

described in ii) above, is on, then the item has been charged out to a user who has not yet picked it up from the library.

The structure of records in the Accession File is illustrated by Figure 7.2.3.1.

Each pointer to a record in the Accession File is three bytes in length and contains the accession number, or record number, of the item concerned. Logical records are blocked into groups of 256 so that the first two bytes of each pointer indicate the number of the physical record that is to be read from the disk and the third byte indicates the logical record within the physical record.

If packing density is defined to be the number of logical records stored per unit of disk space, then calculations based on the track capacity formulas given in IBM [110] for the IBM 2314 Disk Storage Device indicate that this blocking scheme gives a packing density in the disk file approximately 8.4 times greater than if the logical records are unblocked. This means that the Accession File for one million items can be stored in about .52 IBM 2316 Disk Packs instead of about 4.4 IBM 2316 Disk Packs as is required if the records are unblocked. Two main disadvantages accompany the use of such large physical records. The first disadvantage is that extra time is required to transmit the records to and from the disk, which means that the channel can handle fewer requests per unit of time. The second disadvantage is that extra core space is required for the buffers needed to accomodate these large physical records. In particular, if

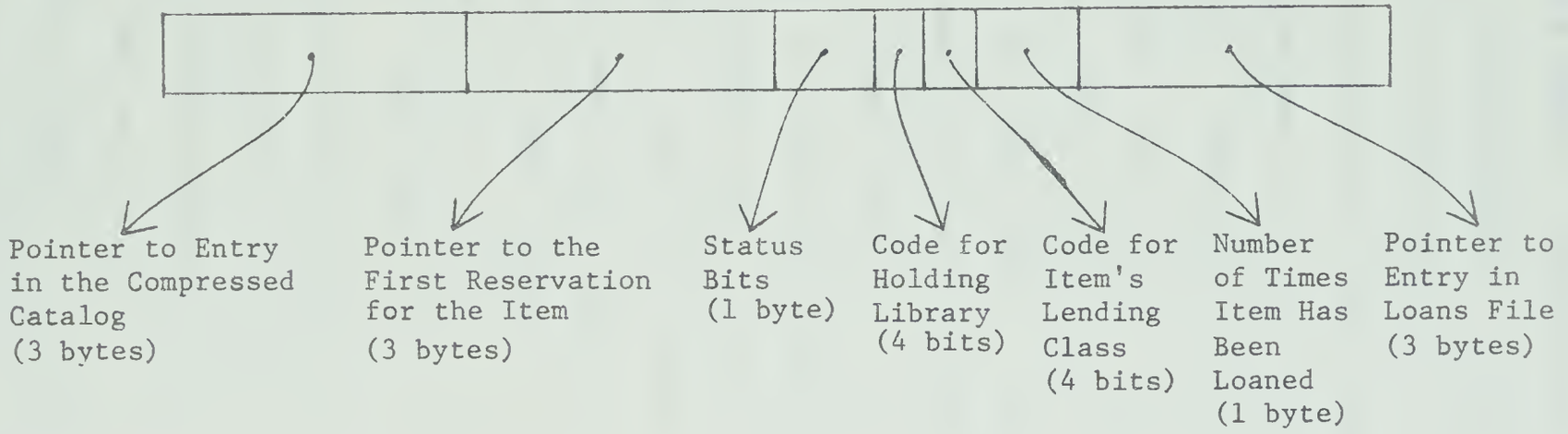


Figure 7.2.3.1 Structure of Entries in the Accession File

the logical records are blocked 256 to a physical record, then each buffer used with the Accession File requires about 3K of core space. This is 256 times larger than if the records are unblocked. In general the optimum block size, or physical record size, is a function of the relative costs and availabilities of core space and disk space in the particular environment in which the library system is to operate. For example, these environmental considerations may indicate that blocking the logical records 128 per physical record yields an overall cost which is lower than if a blocking factor of 256 were used. This blocking factor yields a packing density of 8.1 relative to that obtained with unblocked logical records so that 18.75% more disk space is required, but both the record transmission time and the required buffer size are halved. In addition, the mean access time when the disk and channel are idle is decreased by 7.4% based on a mean seek time of 60 milliseconds (a uniform distribution of seek lengths, or number of cylinders traversed, is assumed) and a rotation time of 25 milliseconds for the IBM 2314. The access time is defined to be the seek time plus the rotational delay plus the record transmission time plus the queueing and channel mask times, if any. If the logical records are unblocked then the mean access time for an idle disk and channel is 14.7% less than that required if the logical records are blocked 256 to a physical record. An additional consideration is that if any blocking factor other than 256 is used, then extra CPU operations are required to translate accession numbers into disk addresses. Some

TABLE 7.2.3.1

Comparison of Blocking Factors for 12-Byte Logical Records[1]

Blocking Factor	1	8	16	32	64	128	256	512
physical record size and buffer size (bytes)	12	96	192	384	768	1536	3072	6144
disk space occupied per physical record (bytes)	114	206	312	523	946	1791	3480	6859
disk space required for 256 logical records (bytes)	29184	6592	4992	4184	3784	3582	3480	3430
relative packing density	1.0	4.4	5.8	7.0	7.8	8.1	8.4	8.5
mean seek time (ms)	60	60	60	60	60	60	60	60
mean rotational delay (ms)	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5
record transmission time (ms)	0.04	0.3	0.8	1.3	2.6	5.3	10.5	21.0
mean access time (ms)	72.5	72.8	73.1	73.8	75.1	77.8	83.0	93.5
mean channel masking time (ms)	12.5	12.8	13.1	13.8	15.1	17.8	23.0	33.5

[1]An IBM 2314 Disk Storage Unit with the Track Overflow Option turned on is assumed.

other blocking factors are compared in Table 7.2.3.1.

7.2.4 The Reservations File

Each entry in the Reservations File is nine bytes in length and contains the following data elements:

i) A three-byte pointer to the Patron File record for the user who placed the reservation.

ii) A two-byte date field which gives the number of the current day relative to some date that has been chosen as an origin.

iii) A two-byte pointer which points to the next entry in the chain of reservations for the reserved book.

iv) A two-byte pointer to the next reservation placed by the patron whose record is pointed to by the contents of the field described in i).

Figure 7.2.4.1 illustrates the structure of entries in the Reservations File.

As is described in Section 7.2.6, each patron's record includes a field that can contain a pointer to the first entry in a list of reservations placed by the patron. The patron's record and the list of reservations form a CORAL ring structure [138] as is illustrated by Figure 7.2.4.2.

Logical records in the Reservations File are blocked 256 per physical record so that the first byte of each pointer to a reservation entry indicates the physical record that is to be read from the disk file and the second byte indicates the logical record within that physical record. This blocking scheme yields a packing density that is 11.4 times greater

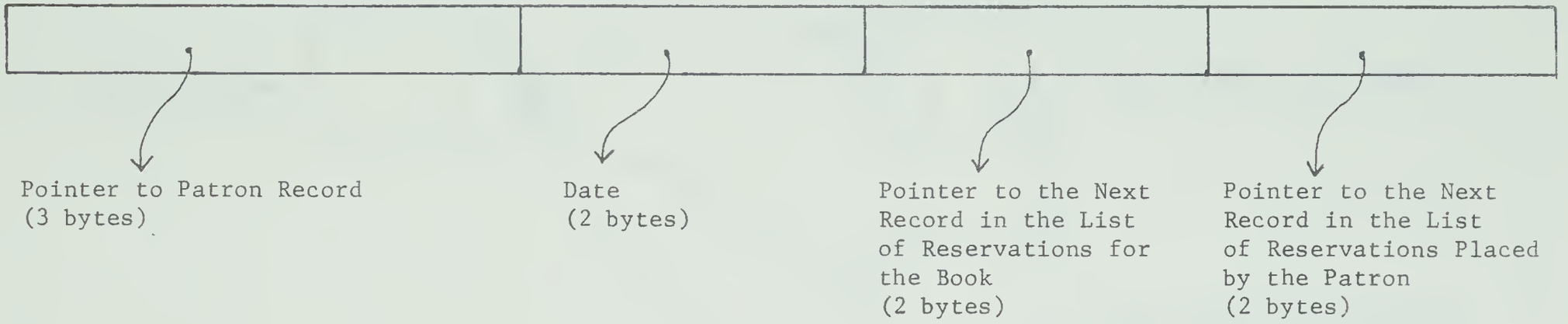


Figure 7.2.4.1 Structure of Records in the Reservations File

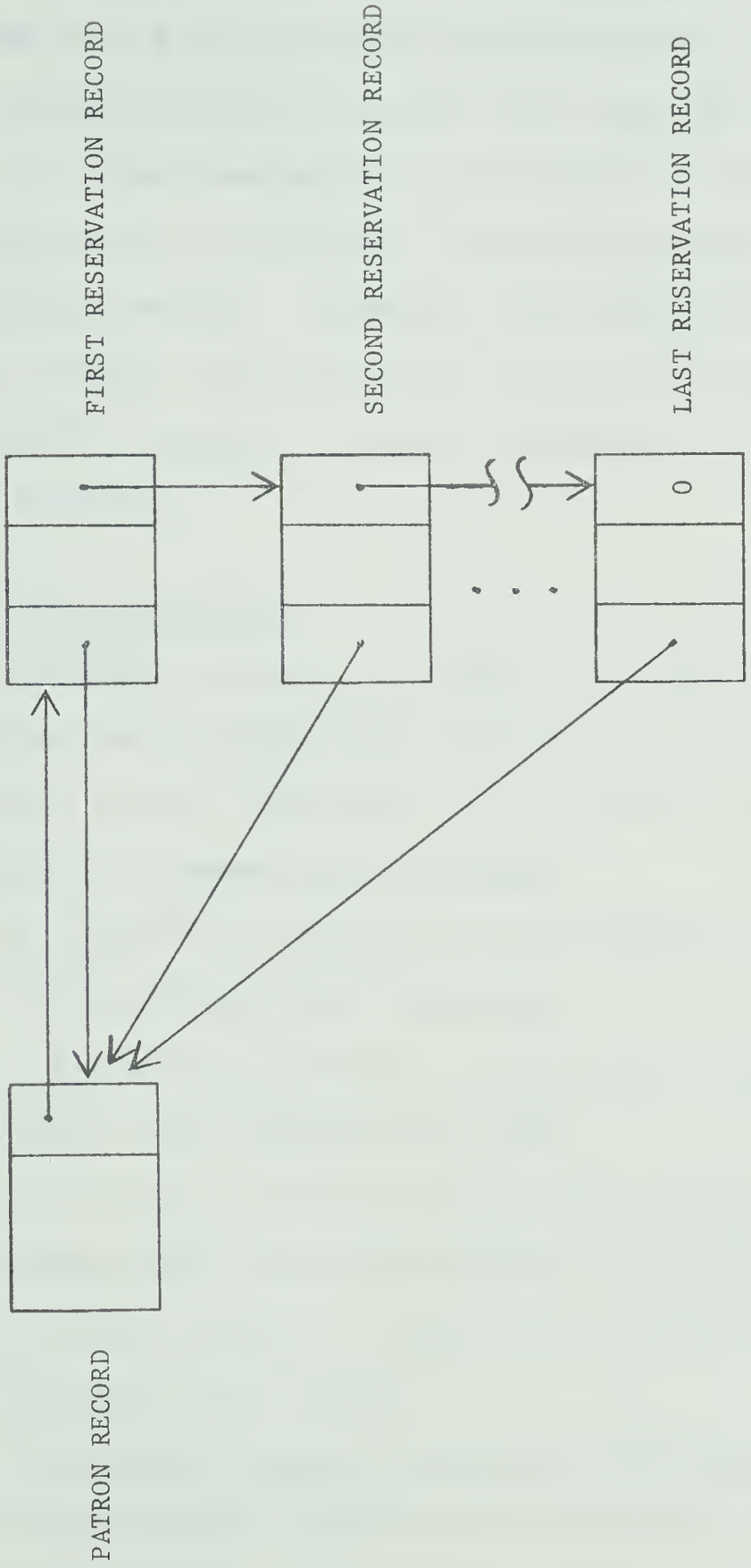


Figure 7.2.4.2 The CORAL Ring Structure That Is Formed by Each Patron's List of Reservations

than that for unblocked records, a buffer size of 2304 bytes, and a mean access time of 80.4 milliseconds, if implemented on an IBM 2314 with the track overflow option turned on. Other possible blocking factors are compared in Table 7.2.4.1.

In actual operation, all records in the Reservations File that are not currently in use are chained into a pool of available records. Whenever a new reservation is entered into the system, the record for it is always drawn from this pool, unless the pool is empty, in which case the Reservations File is extended.

7.2.5 The Loans File

Whenever an item is loaned to a patron a record of the transaction is generated and entered in the Loans File. Each record in the Loans File is 12 bytes in length and contains the following data elements:

i) A three-byte pointer to the accession record of the item that has been borrowed.

ii) A three-byte pointer to the patron record of the patron who borrowed the item.

iii) A two-byte due date field which contains a binary number that gives the number of the day that the item is due relative to a particular date which has been chosen as the origin.

iv) A one-byte transaction status field which contains three one-bit switches that indicate renewal, overdue, and recalled, if on, and a five-bit subfield which contains a code that defines the type and period of

TABLE 7.2.4.1

Comparison of Blocking Factors for 9-Byte Logical Records[1]

Blocking Factor	1	8	16	32	64	128	256	512
physical record size and buffer size (bytes)	9	72	144	288	576	1152	2304	4608
disk space occupied per physical record (bytes)	111	177	252	402	702	1303	2505	4808
disk space required for 256 logical records (bytes)	28416	5664	4032	3216	2808	2606	2505	2404
relative packing density	1.00	5.02	7.05	8.84	10.1	10.9	11.4	11.8
mean seek time (ms)	60	60	60	60	60	60	60	60
mean rotational delay (ms)	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5
record transmission time (ms)	0.03	0.25	0.49	0.99	1.97	3.95	7.88	15.8
mean access time (ms)	72.5	72.7	73.0	73.5	74.5	76.5	80.4	88.3
mean channel masking time (ms)	12.5	12.7	13.0	13.5	14.5	16.5	20.4	28.3

[1]An IBM 2314 Disk Storage Unit with the Track Overflow Option turned on is assumed.

the loan.

v) A three-byte pointer which points to the next loan record for the patron whose patron record number appears in the field described in ii) above. As in the case of records in the Reservations File, unused records in the Loans File are chained into a pool of available records and the file is not extended unless this pool has been exhausted.

Figure 7.2.5.1 illustrates the structure of records in the Loans File.

The logical records of the Loans File are blocked 256 per physical record so that the first two bytes of each three-byte pointer to a logical record in this file indicate the number of the physical record that is to be read from the disk file and the third byte indicates the number of the logical record within the physical record. Thus, pointers to the logical records in this file have the same structure as pointers to logical records in the Accession File, which is described in Section 7.2.3. In addition, Table 7.2.3.1 applies to the Loans File as well as the Accession File.

As is described in Section 7.2.6, each patron's record includes a field that can contain a pointer to the first entry in a list of books borrowed by the patron. The patron's record and the records in the list of borrowed books form a CORAL ring structure [138] as is illustrated by Figure 7.2.5.2.

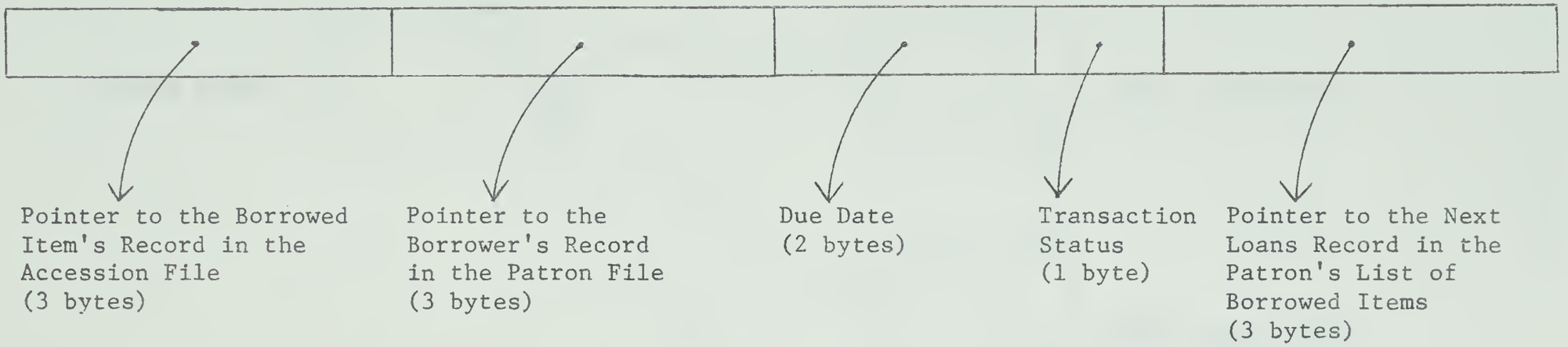


Figure 7.2.5.1 Structure of Records in the Loans File

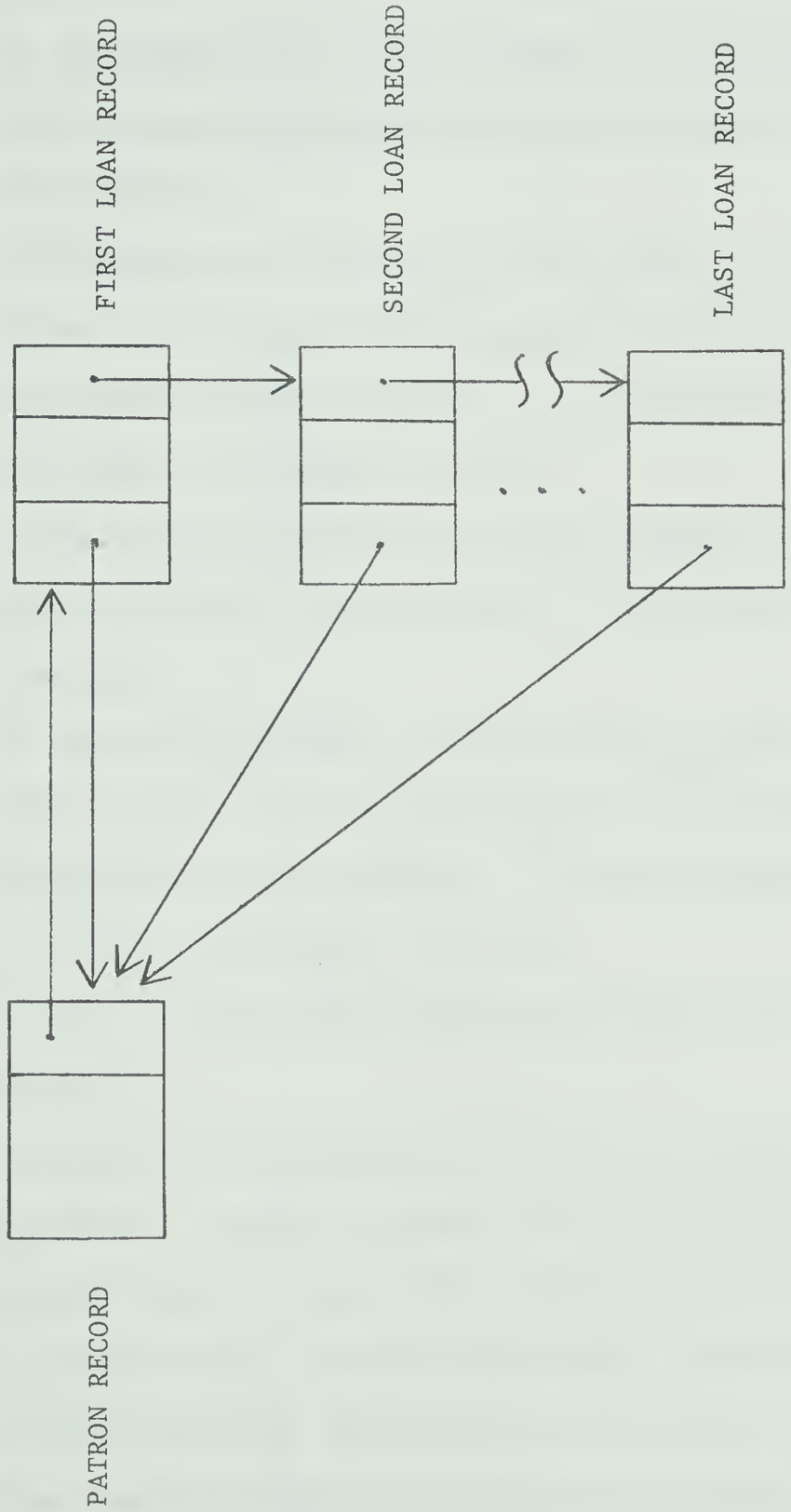


Figure 7.2.5.2 The CORAL Ring Structure That Is Formed by Each Patron's List of Borrowed Books

7.2.6 The Patron File

The Patron File contains one nine-byte record for each library patron. Each record contains the following data elements:

i) A one-byte code that defines the patron's borrowing privileges and the state of the patron's account with the library.

ii) A three-byte pointer to the record in the Loans File that is the first entry in the list of items borrowed by this patron. The structure of this list is shown in Figure 7.2.5.2.

iii) A three-byte pointer to this patron's record in the Master Patron File, which is described in the next section.

iv) A two-byte pointer to the record in the Reservations File that is the first entry in the list of items reserved by this patron. The structure of this list is shown in Figure 7.2.4.2.

Figure 7.2.6.1 illustrates the structure of records in the Patron File.

The logical records in the Patron File are blocked 256 per physical record so that the first 16 bits of each three-byte pointer to this file indicate the number of the physical record that is to be read from the disk file and the last eight bits of each pointer indicate the number of the logical record within the physical record. This blocking factor is compared with several others in Table 7.2.4.1.

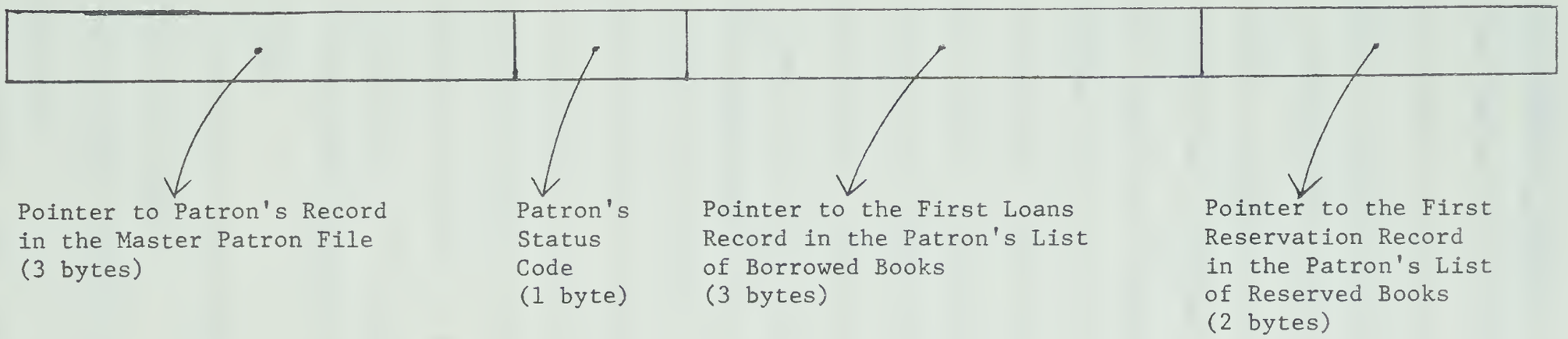


Figure 7.2.6.1 Structure of Records in the Patron File

7.2.7 The Master Patron File

The Master Patron File contains one variable-length record for each of the library's patrons. Each of these Master Patron Records contains the following data elements:

i) A three-byte field that contains the Library Patron Number assigned to the patron. This Library Patron Number is in effect a pointer to the Patron File record for the patron because it is identical to the relative number of the patron's record in the Patron File.

ii) A four-bit field that contains a code which defines the title of the patron.

iii) A four-bit field that contains a code which defines the patron's borrowing privileges.

iv) A one-byte field that contains a code which defines the department or faculty to which the patron belongs.

v) A three-byte field that contains the telephone number of the patron, encoded as a binary number.

vi) A four-byte field that contains the student identification number or the social insurance number of the patron, encoded as a binary number. The relative magnitude of the number indicates which case applies.

vii) A variable-length field that contains the patron's name.

viii) A one-byte field that indicates the length of the patron's name.

ix A variable-length field that contains the patron's address.

x) A one-byte field that indicates the length of the

patron's address.

The structure of these Master Patron Records is illustrated by Figure 7.2.7.1.

The Master Patron Records may be mapped into the physical disk file by either of two different methods. In the first method, 16 variable-length logical records are grouped into each variable-length physical record as is illustrated by Figure 7.2.7.2. If it is assumed that the minimum length of the patron name field is five bytes and that the minimum length of the patron address field is ten bytes, then it may be assumed that the minimum length of a master patron record is 28 bytes. The minimum length of a physical record, may then be taken as 28 x 16, or 448, bytes plus the 18 bytes of the directory portion of the physical record, or 464 bytes in total. Calculations based on the track capacity formulas given in IBM [110] indicate that no more than 13 of these minimum-length physical records can fit into one track of an IBM 2314. Therefore, four bits are sufficient to number the physical records on one track. Hence, each three-byte pointer to the Master Patron File is structured so that the first two bytes indicate the relative track within the file, the next four bits indicate the physical record within the track, and the last four bits indicate the logical record within the physical record. Figure 7.2.7.3 illustrates the structure of these pointers. This pointer structure would still apply if each logical record were prefixed by its length instead of having all the lengths grouped together into a directory.

The second method of mapping the variable-length

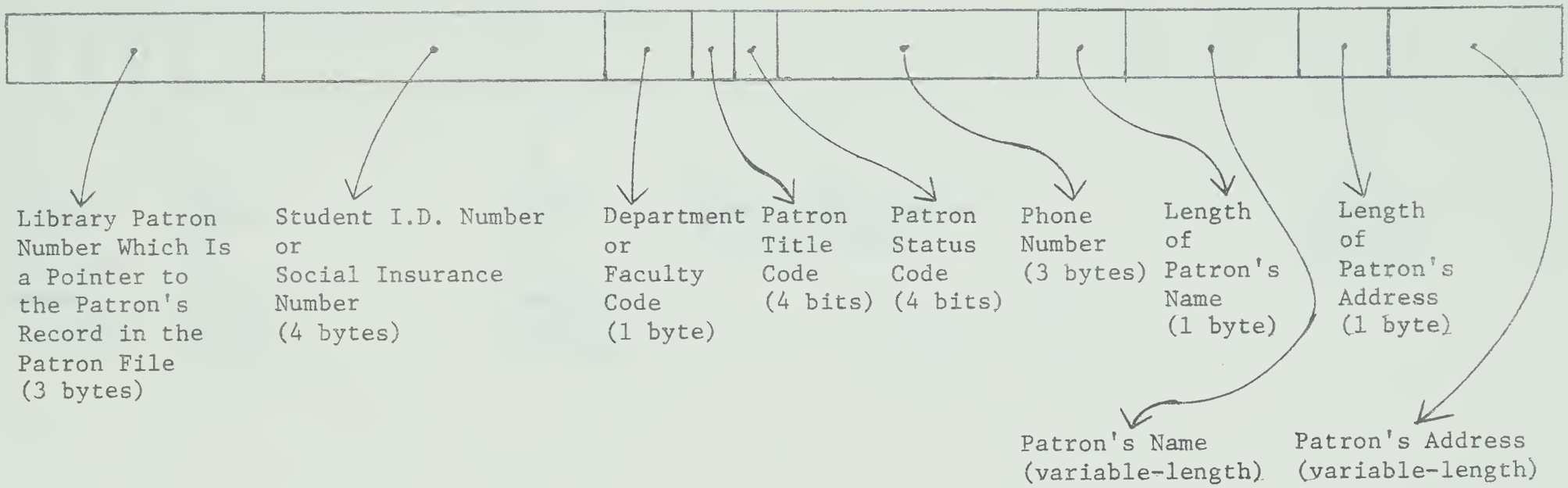


Figure 7.2.7.1 Structure of Records in the Master Patron File

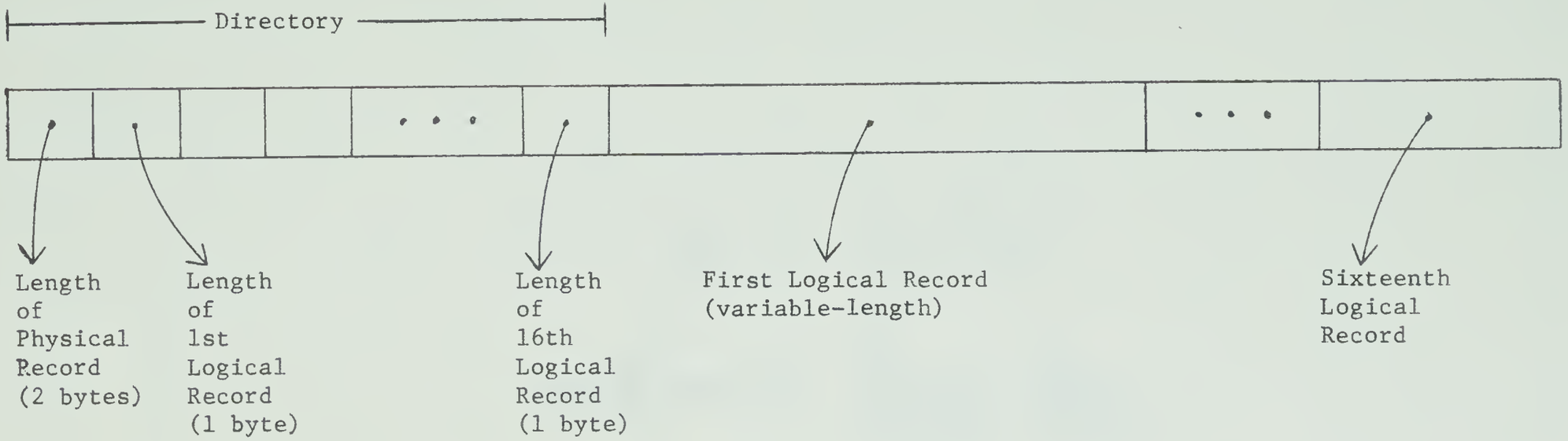


Figure 7.2.7.2 Structure of Physical Records in Master Patron File When the Logical Records Are Blocked 16 per Variable-Length Physical Record

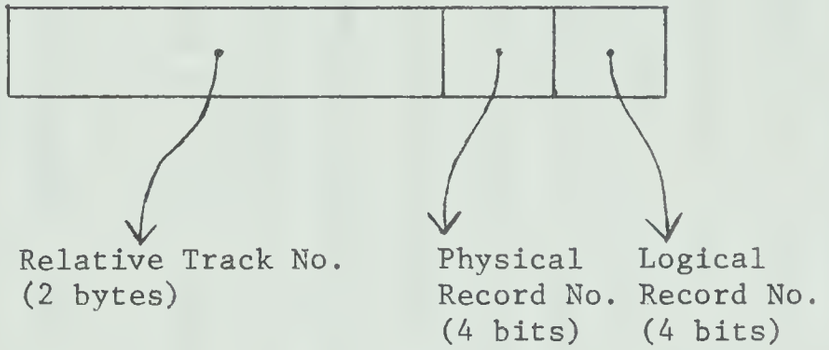


Figure 7.2.7.3 Structure of Pointers to the Master Patron File When the Scheme Illustrated in Figure 7.2.7.2 Is Used

logical records of the Patron File into physical records is shown in Figure 7.2.7.4. This method is based on a similar one outlined by J. L. Cunningham et al. [78]. Each of the physical records is of a fixed size and has a variable number of the variable-length logical records mapped into it. If necessary, the last logical record in a physical record may overflow into the next physical record. In this scheme, each four-byte pointer to a record in the Master Patron File is structured so that the first 15 bits give the number of the physical record that contains the required logical record, the next 10 bits give the starting position of the logical record within the physical record, and the last seven bits give the length of the logical record.

This second pointer structure implies that the Master Patron File may contain up to 2^{15} or 32,768 physical records each of which is 2^{10} or 1024 bytes in length and that logical records may be up to 2^{7} or 128 bytes in length.

If it is assumed that the average Master Patron Record is 64 bytes in length, then each physical record may be assumed to contain an average of 1024/64 or 16 logical records; and if it is assumed that the average logical record is 32 bytes in length, then each physical record may be assumed to contain an average of 1024/32 or 32 logical records. Therefore, the Master Patron File can be expected to accomodate from $2^{4} \times 2^{15}$ or 2^{19} patron records up to $2^{5} \times 2^{15}$ or 2^{20} patron records.

Since the information contained in the Master Patron

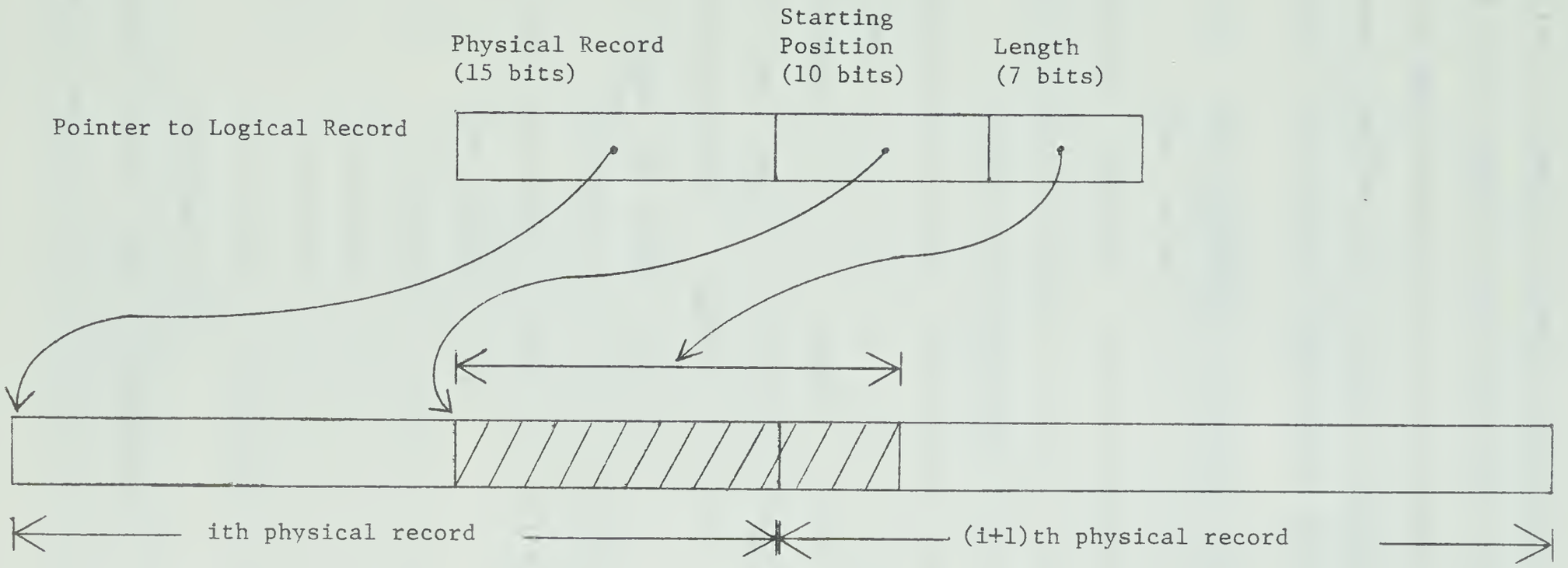


Figure 7.2.7.4 Mapping the Variable-Length Master Patron Records into Fixed-Length Physical Records

File is likely to have a relatively low frequency of use and is also likely to be used almost exclusively in batch processing operations such as the preparation of overdue notices, recall notices, and fines statements, it may be desirable to store this file on tape rather than on disk.

7.2.8 The Accession Directory File

The records of the Accession Directory File are used, whenever necessary, to link records in the Accession File to their parent records in the LC File.

Every four-byte entry, or logical record, in the Accession Directory File may be used in either of two ways. First, it may point to a set of contiguous entries in the Accession File; or, second, it may point to a set of contiguous entries in the Accession Directory File itself. Every directory entry contains a one-bit flag that indicates which of these two cases applies to it.

In the first case, illustrated by Figure 7.2.8.1, a directory entry contains the following data elements:

i) The one-bit flag.

ii) A three-byte pointer to the first entry of a set of contiguous entries in the Accession File.

iii) A seven-bit field that gives the number of entries in the set of accession entries pointed to by the pointer field.

In the second case, illustrated by Figure 7.2.8.2, a directory entry contains the following data elements:

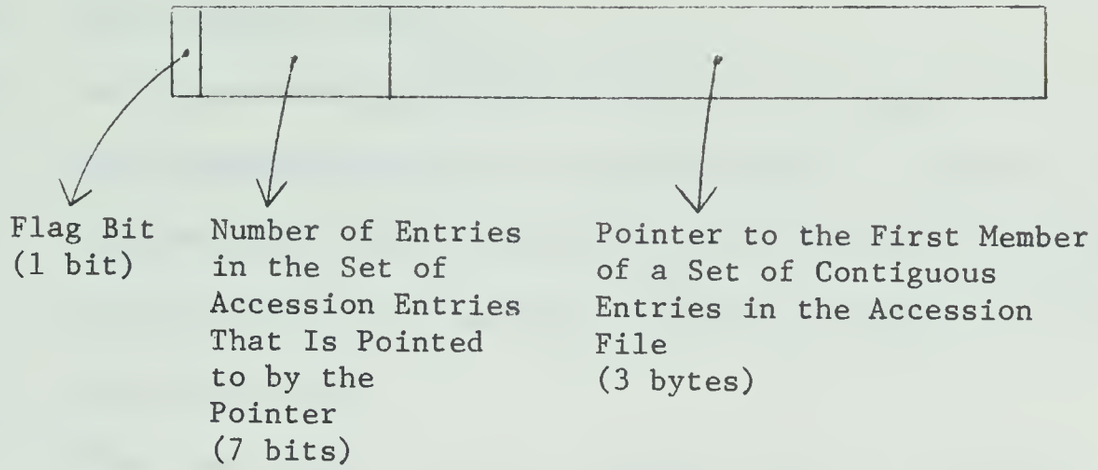


Figure 7.2.8.1 Structure of Accession Directory Records: Case One

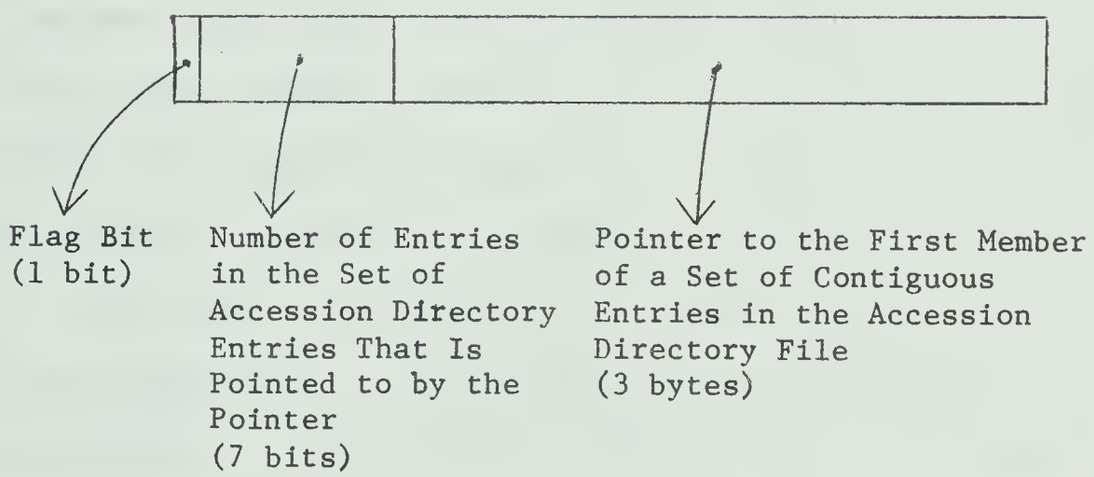


Figure 7.2.8.2 Structure of Accession Directory Records: Case Two

i) The one-bit flag.

ii) A three-byte pointer to the first entry of a set of contiguous entries in the Accession Directory File.

iii) A seven-bit field that gives the number of entries in the set of directory entries pointed to by the pointer field.

The four-byte logical records of the Accession Directory File are blocked 256 per physical record. Hence, in each three-byte pointer to a logical record in this file, the first two bytes give the number of the physical record that contains the required logical record and the third byte gives the number of the logical record within the physical record. This scheme yields a buffer size of 1024 bytes, a mean access time for an idle disk and channel of 76.0 milliseconds, and a packing density of 21.5 relative to that obtained with unblocked four-byte records. Some other possible blocking factors are compared in Table 7.2.8.1.

Now that the records of the Accession Directory File have been described, it is appropriate to describe their use in more detail. This will be done, in turn, for the cases of monographs, closed multi-volume sets, open multi-volume sets, and bound and unbound journals.

In the case of monographs, directory entries are used to link accession records for original copies and added copies to their parent record in the LC File. Whenever a new monograph is received by the library, all copies of it are allocated consecutive entries in the Accession File and a pointer to the first member of this set of contiguous

TABLE 7.2.8.1

Comparison of Blocking Factors for 4-Byte Logical Records[1]

Blocking Factor	1	8	16	32	64	128	256	512
physical record size and buffer size (bytes)	4	32	64	128	256	512	1024	2048
disk space occupied per physical record (bytes)	105	136	171	241	382	664	1227	2353
disk space required for 256 logical records (bytes)	26380	4352	2736	1928	1528	1328	1227	1176
relative packing density	1.00	6.05	9.50	13.7	17.2	19.8	21.5	22.4
mean seek time (ms)	60	60	60	60	60	60	60	60
mean rotational delay (ms)	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5
record transmission time (ms)	0.02	0.11	0.22	0.44	0.88	1.76	3.51	7.02
mean access time (ms)	72.5	72.6	72.7	72.9	73.4	74.3	76.0	79.5
mean channel masking time (ms)	12.5	12.6	12.7	12.9	13.4	14.3	16.0	19.5

[1]An IBM 2314 Disk Storage Unit with the Track Overflow Option turned on is assumed.

entries, along with the number of entries in this set, is installed in the LC File entry for the new monograph. This situation is illustrated by Figure 7.2.8.3.

If the library later acquires additional copies of the monograph, then the added copies will likewise be assigned consecutive entries in the Accession File; but the set of accession entries for the added copies is not located adjacent to the set of accession entries for the original copies. The single Accession File Pointer contained in the LC File entry for the monograph cannot simultaneously point to both sets of accession records since they are located in discontinuous regions of the Accession File, the intervening entries being assigned to other titles.

This difficulty can be circumvented by using a pair of adjacent directory entries which are pointed to by the pointer in the LC File entry. These directory entries, in turn, contain the actual pointers to the two nonadjacent sets of entries in the Accession File. The number of copies field in the LC File entry is now used to indicate the number of directory entries in the set pointed to by the pointer. This situation is illustrated by Figure 7.2.8.4.

If at a later date, the library were to acquire a second set of added copies of the monograph then a new set of three contiguous directory entries would be assigned to the monograph. The contents of the two old directory entries would be moved to the first two entries of the new set and a

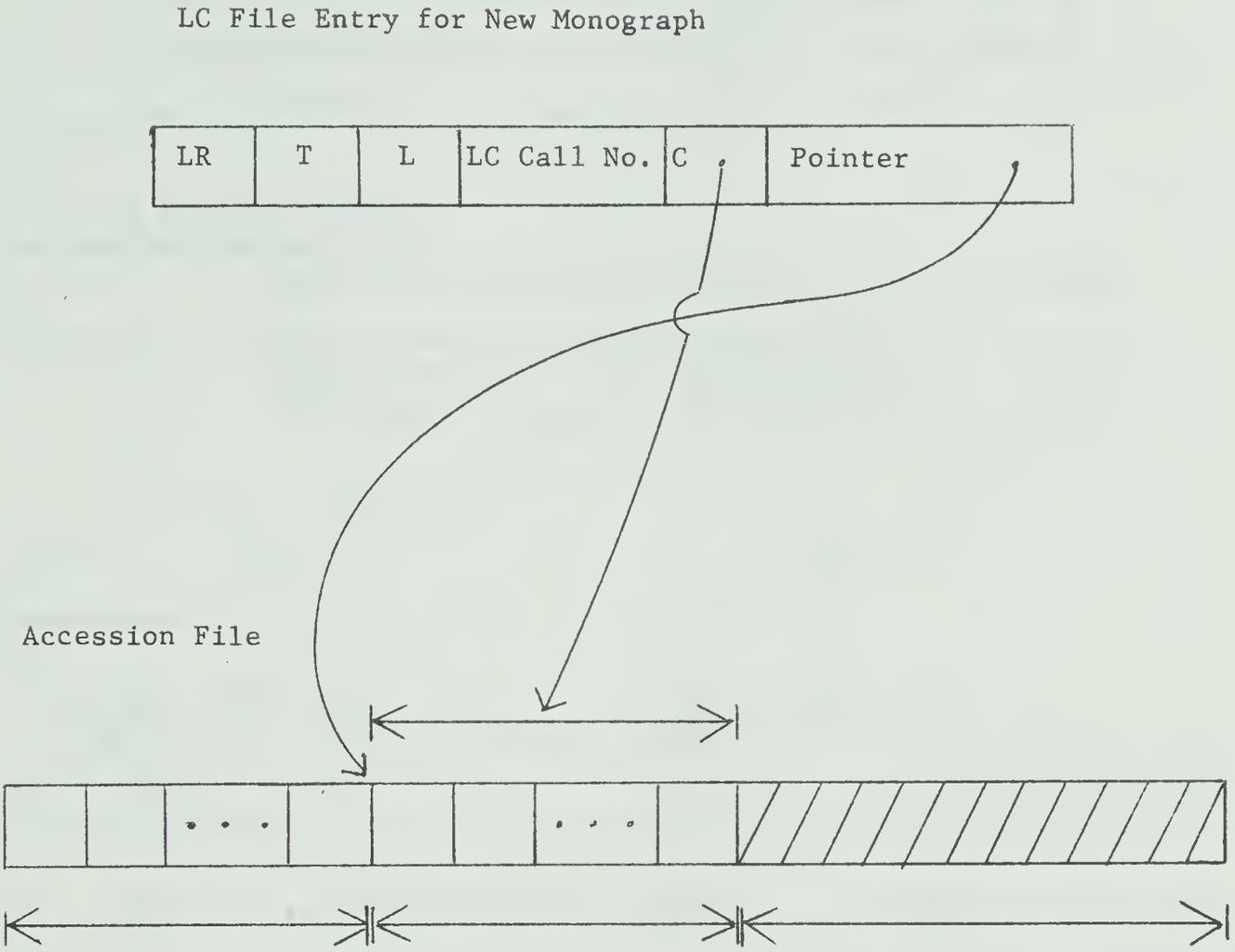


Figure 7.2.8.3 Linkage Setup Between Accession File and LC File for the Case of Monographs with No Added Copies

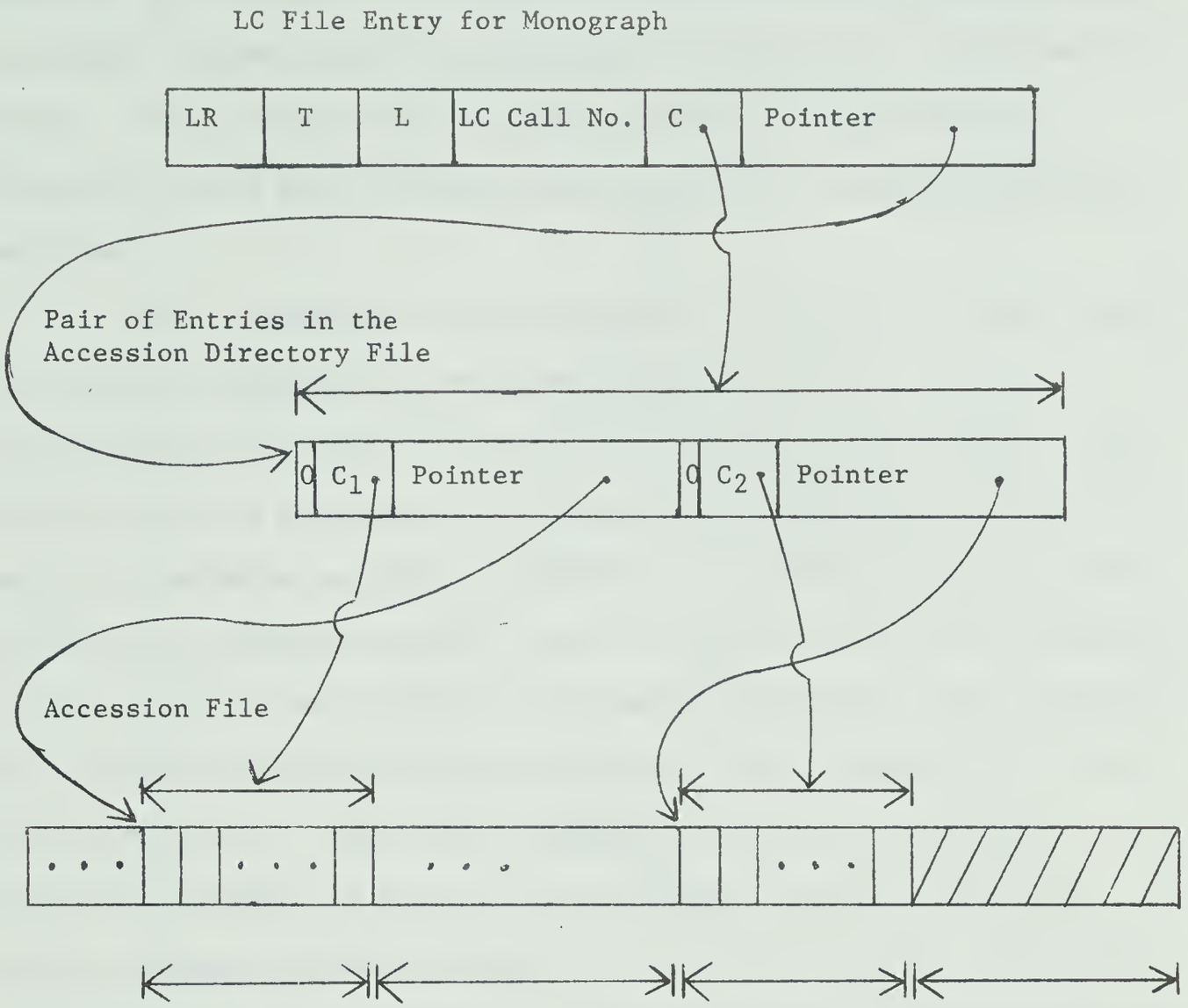


Figure 7.2.8.4 Linkage Setup Between Accession File, Accession Directory File, and LC File for the Case of Monographs with Added Copies

pointer to the set of accession records for the second group of added copies would be installed in the third new directory entry. The old pair of directory entries would then be chained into a pool of contiguous pairs of vacated directory entries.

It is apropos at this point to describe in more detail the pooling scheme for vacated directory blocks. As is illustrated in Figure 7.2.8.5, this scheme uses a pool table which contains pointers to the starting members of chains of pairs of vacated entries, triplets of vacated entries, quadruplets of vacated entries, and so forth, as well as pointers to the end of the currently allocated space for the directory file and to the end of the currently occupied portion of the directory file. When, for example, a pair of directory entries is added to the pool of vacated entries, the pool table, normally disk resident, is read into core. Then the current pointer to the start of the chain of vacated pairs is inserted into the pointer field of the first member of the new pair which is then written out onto the disk. A pointer to this new pair is then installed in the pool table which is then written back onto the disk. Whenever a new set of directory entries is required the pool table is read into core and searched to see if a vacant set of entries of the required size exists. If one does exist, then it is used in preference to extending the occupied portion of the directory file.

Now that the description of the pooling scheme for the

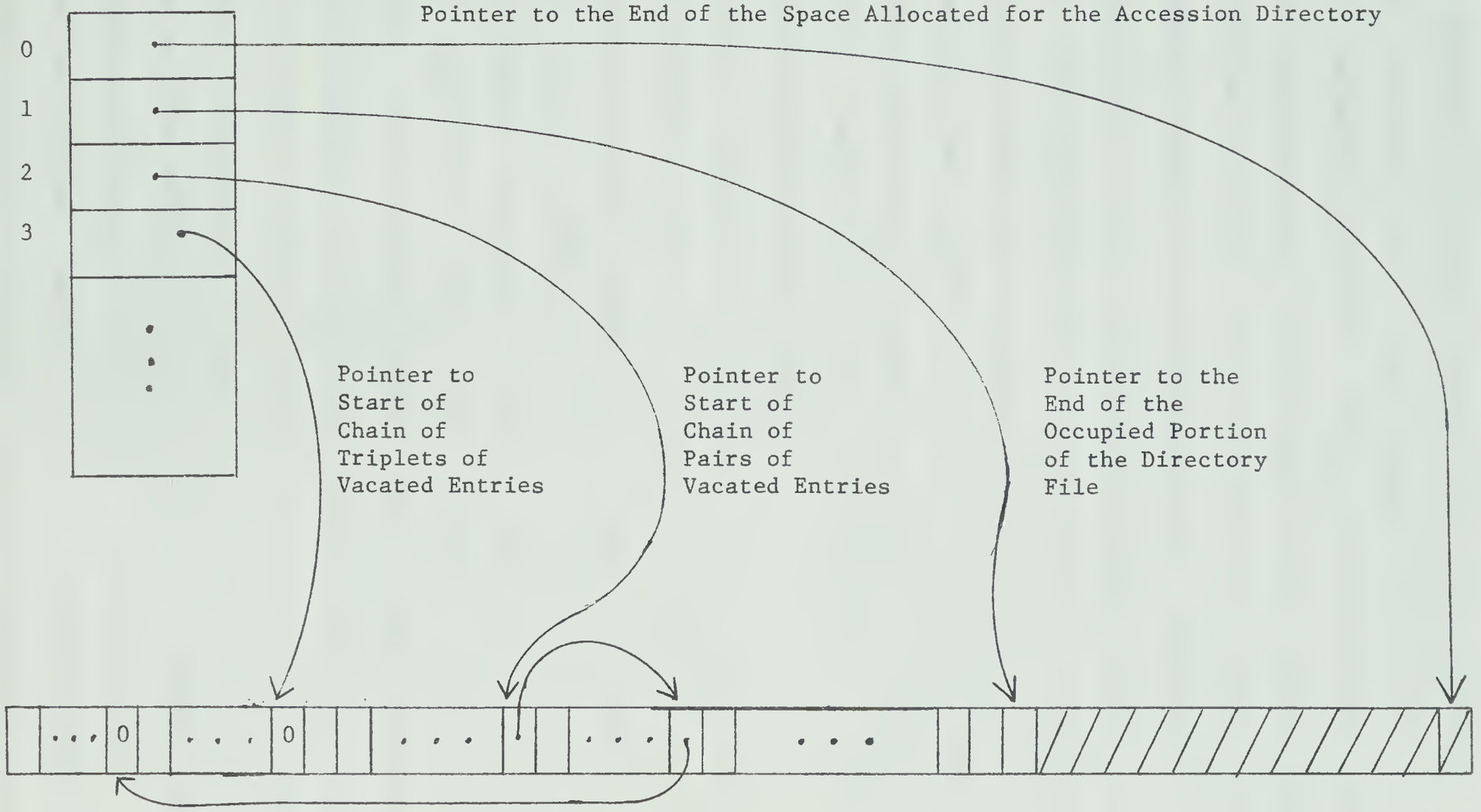


Figure 7.2.8.5 The Pooling Scheme for Vacated Accession Directory Entries

vacated accession directory records has been completed, it is proper to return to the description of the use of the accession directory records. The application of the directory records to both open and closed multi-volume sets is essentially similar. In both cases, the directory entries link the accession records of all copies of all volumes of a particular title to the corresponding parent record in the LC File. The only difference is that the set of directory entries for an open multi-volume set must continuously expand to accommodate new volumes as they are received by the library. In contrast, the set of directory entries for a closed multi-volume set is essentially static, provided that no added copies are acquired by the library.

As is illustrated in Figure 7.2.8.6, the LC File entry for either an open or a closed multi-volume set contains a pointer to the first member of the group of accession directory entries associated with it. This group of directory entries contains one entry for each volume of the multi-volume set, the ith directory entry being assigned to the ith volume of the set. If the library should acquire added copies of a particular volume of the set, say volume V, then the directory entry for volume V would be changed so that it would point to a second level group of directory entries which would in turn contain the actual pointers to both the original and added copies of volume V of the set. The second level directory entries are "second level" in a logical sense only since they are still contained within the Accession Directory

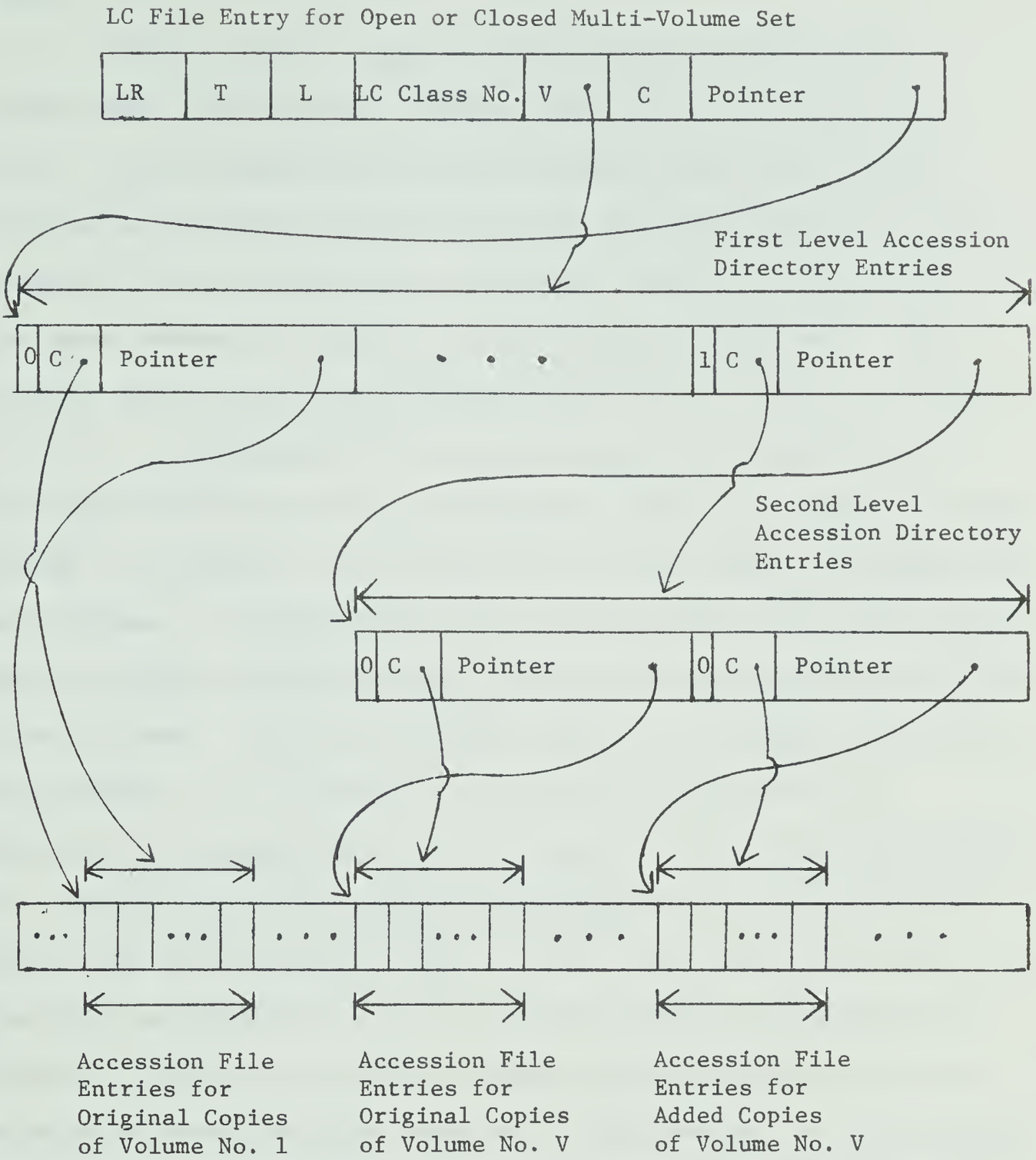


Figure 7.2.8.6 Linkage Setup Between Accession File, Accession Directory, and LC File for the Cases of Open and Closed Multi-Volume Sets

File.

There exists a special case of the closed multi-volume set that does not require the use of the Directory File. In this special case, the library has acquired all volumes of the set at the same time, no added copies of any volume of the set have been obtained, and the library has the same number of copies of each volume of the set. The linkage scheme for this special situation is shown in Figure 7.2.8.7. It depends on the allocation of a group of V x C contiguous Accession File entries for the set, where V is the number of volumes in the set and C is the number of copies of each volume. The pointer in the LC File entry for the title points to the accession entry for the first copy of the set's first volume. Given the volume number, or volume number and copy number, it is possible to directly calculate the address of the required entry or set of entries in the Accession File. Two advantages are provided by the use of this method for this special case. First, one less disk access is required as compared to the standard method illustrated by Figure 7.2.8.6; and, second, space is not required for the storage of the directory entries. Furthermore, if the library later requires added copies of one or more volumes of the set, then the standard method is reverted to easily enough by simply inserting the required directory entries.

The use of the Accession Directory File with journals is basically identical to its use with open multi-volume sets except for the added complication that unbound issues of the current volume and of the volume just prior to the current

LC File Entry for Closed Multi-Volume Set

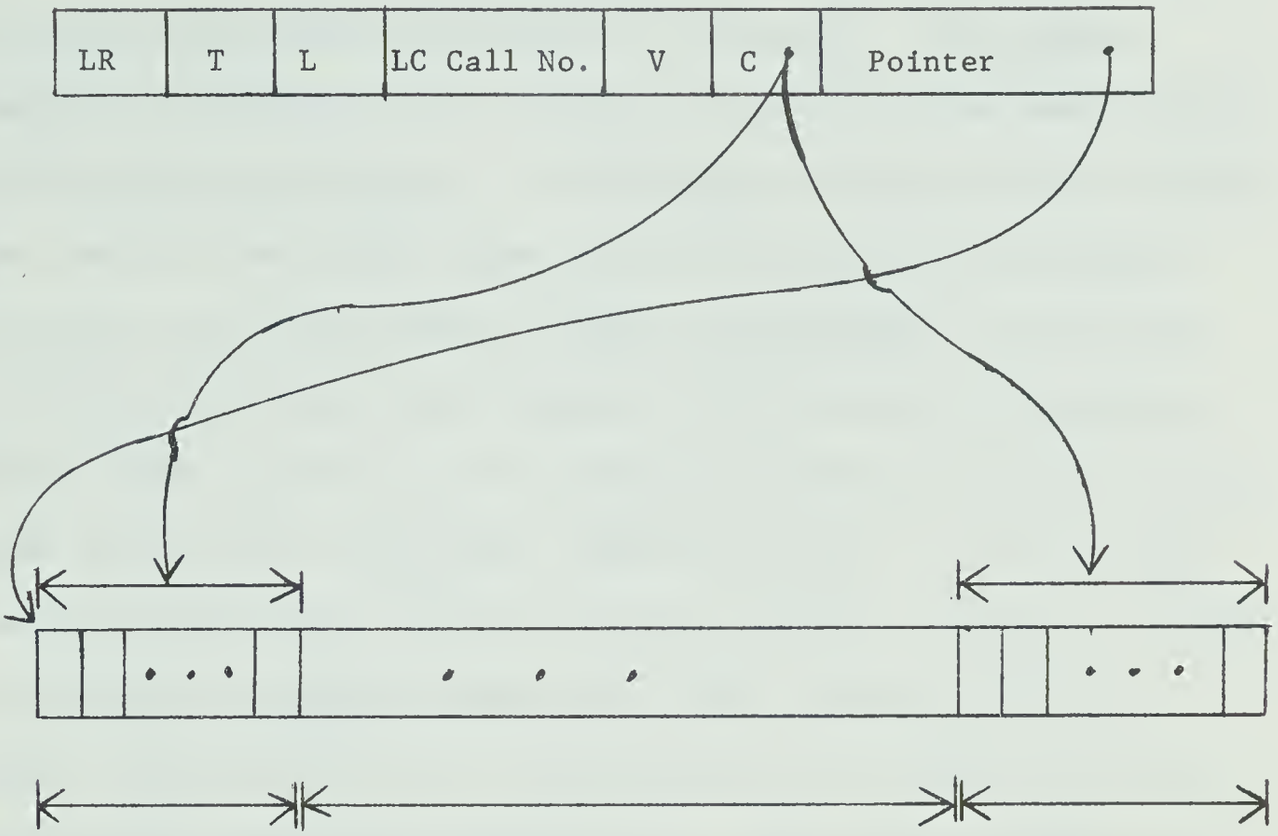


Figure 7.2.8.7 Linkage Setup Between the Accession File and the LC File for the Special Case of a Closed Multi-Volume Set with the Same Number of Copies in Each Volume and No Added Copies

volume must be handled in addition to the bound volumes of the journal.

Figure 7.2.8.8 illustrates the scheme employed for both bound and unbound volumes of a journal. The bound volumes of the journal are handled in exactly the same manner as a multi-volume open set. The unbound issues of the current volume and of the volume just prior to the current volume, also called the old current volume, are handled as follows. The LC File entry for the journal title contains a pointer, labelled APTR, to the set of accession records for the current unbound volume and a pointer, labelled APTR', to the set of accession records for the old current volume. When all issues of the current unbound volume have been received by the library, the counter V_c is incremented by one and the information in the locations labelled N,C, and APTR is moved to the locations labelled N',V', and APTR' so that locations N,C, and APTR may be used by the new current volume. At this time steps are taken to bind the issues of the old current volume. First, a contiguous block of Accession File records, large enough to accomodate the C' bound copies of the old current volume, is obtained. Second, a new set of accession directory entries which contains one more member than the old set is obtained as described earlier for the case of monographs. Then, the contents of the old set are moved to the first V_b entries of the new set, and a pointer to the newly acquired set of accession records for bound volume $V_b + 1$, the old current volume, is installed in the last entry of the

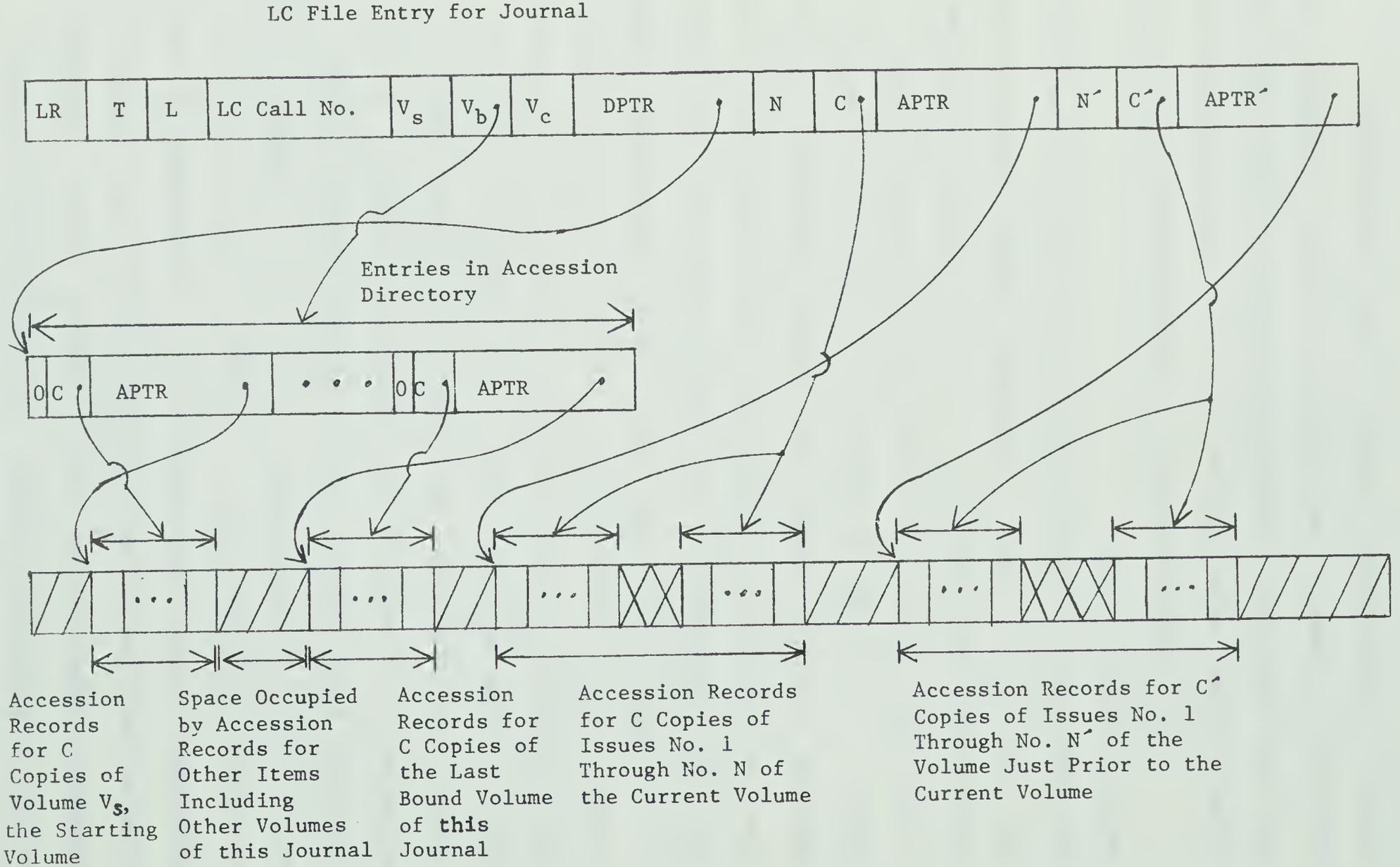


Figure 7.2.8.8 Linkage Setup Between Accession File, Accession Directory, and LC File for the Case of Journals

new set of directory entries. The old set of directory entries is then returned to the pool of vacated accession directory entries as was described earlier for the case of monographs. Third, the counter V_b is incremented by one and the pointer labelled DPTR is changed so that it points to the new set of directory entries. Fourth, as a complete set of issues for each copy of the old current volume becomes available it is sent to the bindery to be bound. If the library holds more than one copy of the journal then only one copy at a time would be sent to the bindery so that the remaining copy or copies would be available to the library's patrons. Fifth, as soon as all copies of the old current volume of the journal have been bound, the accession records which were dedicated to its unbound issues may be returned to a pool of accession records which would be managed in a manner identical to that employed with vacated accession directory records, or they may be reserved for the use of unbound issues of the volume that is to come after the current volume. The latter method would provide a form of "exchange buffering" since the same two sets of accession records would be used many times over, alternatively as the current volume and as the old current volume.

The foregoing method of handling periodicals' circulation records, when combined with the On-line Catalog System, forms the basis for a computerized serials system that could automatically handle the receiving, routing, claiming, and binding of periodicals as well as their circulation.

CHAPTER VIII

CONCLUSION

The investigation reported in this thesis was composed of two principal parts. The first part involved the design, implementation, test, and evaluation of an integrated library automation system that encompassed an on-line catalog subsystem and a real-time circulation subsystem. This pilot system, the IT system, was intended for use in a small, specialized, academic library.

The primary purposes to be achieved by the implementation and operation of this pilot system were as follows:

i) To demonstrate the feasibility of implementation of an on-line integrated library automation system.

ii) To test and evaluate certain preconceived notions about the structure of the command language, and about the requirements that must be met if such an on-line integrated automation system is to serve the needs of a small specialized library.

iii) To discover any inadequacies in the scope, design, or implementation of the pilot system and its supporting file structures.

iv) To obtain experience that would be applicable to the design of a more advanced system that would serve the needs of a large academic library.

These goals were achieved. The final version of the pilot

system is described in Chapters III, IV, and V.

Experience in the use of the IT system has demonstrated that the commands are easy to remember and that the system is generally convenient to use. To an experienced user some of the command formats and some of the messages typed by the system appear to be more detailed than necessary and should be abbreviated to reduce the time needed to enter the commands and to wait for the typing of messages to be completed. If a CRT terminal were used instead of a typewriter terminal this objection would no longer apply to the output of messages by the system.

Those who have used the system have criticized the fact that multiple entries for loans, requests, returns, and renewals are not possible with the system described in this treatment. It has been remarked that following a LOAN command it would be convenient to enter a list of separate items without having to repeat the command for each entry. The system has, in fact, been modified to allow such multiple entries.

In general the command structure of IT has proved to be very satisfactory. It is convenient for demonstration purposes, it may be easily learned, and it requires only slight modifications to make it satisfy the requirements of an experienced operator.

The second part of the investigation involved the design of a file structure that could support an on-line integrated library automation system that would serve the

needs of a large academic library. The proposed file structure outlined in Chapters VI and VII is confined to the files that are necessary for the support of an on-line catalog subsystem and a real-time circulation subsystem. It represents an attempt to design a file structure that is economical in terms of storage requirements and in terms of CPU time needed to maintain and search it. The file structure must be capable of supporting a wide range of on-line activities and of providing very short response times for most on-line transactions and queries.

The file structure for the real-time circulation subsystem was specified in greater detail than the file structure for the on-line catalog subsystem. This was because much more is known about the requirements that must be met by a real-time circulation system and its supporting file structure. For example, comparatively little is known about the catalog search strategies employed by library patrons, the structure of the user-system interface, the frequencies of occurrence of words in the various fields of the catalog entry, the frequencies of occurrence of terms in queries, and the frequencies of occurrence of non-standard versus standard terms in queries. Consequently, the specifications for the proposed file structure for the on-line catalog are restricted in scope and detail.

Most of the effort in this phase of the investigation was expended in the design of a new method for the construction of the index files for the on-line catalog. The proposed

method, which is based on the principles of virtual hash addressing, is described in detail in Chapter VI. The description of this method covers the detailed structures of the hash index files for author, title words, and LC call numbers, the procedures to be used in the generation, maintenance, and search of these files, and the theoretical expected performance of these files in terms of file storage requirements and file access times.

It is believed that the proposed method for the construction of the indexes to the on-line catalog can provide extremely rapid access to the entries in these indexes while requiring less disk storage than that required by conventional methods. In addition, the hash index files integrate well with the term dictionaries that are needed to encode and decode the entries of the proposed compressed catalog file. However, the performance of these hash index files is still largely a matter of conjecture, since little experimentation has been carried out with them. An experimental comparison of the proposed method with the traditional methods of index construction should be undertaken to confirm or reject the hypothesized superior performance of the proposed method.

In addition to the problems associated with the provision of rapid access to the entries of the indexes to the on-line catalog, the design of a supporting file structure for the on-line catalog involves problems associated with the provision of a thesaurus that would aid in the search of the on-line catalog, problems associated with the storage and

maintenance of the compressed bit strings files, and problems associated with the structure of the compressed catalog file itself. In general, conventional designs of computer file structures are inadequate for the on-line maintenance and search of bibliographic data bases that contain a million or more entries. Consequently, there is a need for continuing research and innovation before large bibliographic files and their indexes will be searched efficiently from on-line terminals.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Kent, A. and Lancour, H. (eds.), Encyclopedia of Library and Information Science, Vol. 2, Marcel Dekker, Inc., New York, p. 211.

2. Howe, M. T., "The Establishment and Growth of the Data Processing Department in the Decatur Public Library", Data Processing in Public and University Libraries, Edited by J. Harvey, Sparten Books, Washington, 1966, pp. 37-52.

3. Griffen, M., "IBM Advanced System Development Library in Transition", Proceedings of the 1963 Clinic on Library Applications of Data Processing, Edited by H. Goldstein, University of Illinois, 1964, pp. 79-95.

4. Parker, R. H., "Development of Automatic Systems at the University of Missouri Library", Proceedings of the 1963 Clinic on Library Applications of Data Processing, Edited by H. Goldstein, University of Illinois, 1964, pp. 43-61.

5. Griffen, H. L., "Electronic Data Processing Applications to Technical Processing and Circulation Activities in a Technical Library", Proceedings of the 1963 Clinic on Library Applications of Data Processing, Edited by H. Goldstein, University of Illinois, 1964, pp. 96-108.

6. Kraft, D. H., "A 'Total Systems' Approach to Library Mechanization", Proceedings of the First Texas Conference on Library Mechanization, Edited by J. B. Corbin, Texas Library and Historical Commission, 1966, pp. 7-12.

7. Heaps, D. M. and Heaps, H. S., "Report on a Program for Library Automation", Department of Computing Science, University of Alberta, 1968.

8. Jennings, M. A., "Optimizing Library Automation with a Central Dynamic Store", College and Research Libraries, Vol. 30, No. 5 (September 1969), pp. 397-404.

9. Kilgour, F. G., "Comprehensive Modern Library Systems", The Brasenose Conference on the Automation of Libraries, Edited by J. Harrison and P. Laslett, Mansell Information, 1967, pp. 46-56.

10. Warheit, I. A., "Design of Library Systems for Implementation with Interactive Computers", Journal of Library Automation, Vol. 3, No. 1 (March 1970), pp. 65-78.

11. Alanen, S., Sparks, D. E., and Kilgour, F. G., "A Computer-Monitored Library Technical Processing System", Proceedings of the ADI Annual Meeting, Vol. 3, Adrianne Press, 1966, pp. 419-426.

12. Hayes, R. M., "Concept of an On-Line Total Library System", Library Technology Reports, American Library Association, Chicago, 1965.

13. Schultheiss, L. A., Culbertson, D. S., and Heiliger, E. M., Advanced Data Processing in the University Library, The Scarecrow Press, New York, 1962.

14. Parker, R. H., "Library Records in a Total System", The Brasenose Conference on the Automation of Libraries, Edited by J. Harrison and P. Laslett, Mansell Information, 1967, pp. 32-45.

15. Griffin, M., "Automation in Libraries: A Projection", Canadian Library, Vol. 23, No. 5 (March 1967), pp. 360-367.

16. Payne, C. T., "An Integrated Computer-Based Bibliographic Data System for a Large University Library: Problems and Progress at the University of Chicago", Proceedings of the 1967 Clinic on Library Applications of Data Processing, Edited by D. E. Carroll, University of Illinois, 1967, pp. 29-40.

17. Hammer, W. C., "Mechanized Circulation Control", Practical Problems of Library Automation, Documentation Group, Washington, D. C. Chapter, Special Libraries Association, 1968, pp. 33-37.

18. Wilkinson, W. A., "The Computer-Produced Book Catalog: An Application of Data Processing at Monsanto's Information Centre", Proceedings of the 1964 Clinic on Library Applications of Data Processing, Edited by H. Goldstein, University of Illinois, 1965, pp. 7-24.

19. Bregzis, R., "The ONULP Bibliographic Control System", Proceedings of the 1965 Clinic on Library Applications of Data Processing, Edited by F. B. Jenkins, University of Illinois, 1966, pp. 112-140.

20. Matthews, F. W., and Oulton, D. L., "A Simplified Computer Produced Book Catalogue", Proceedings of the American Documentation Institute, Vol. 4, Thompson Book Company, Washington, 1967, pp. 191-196.

21. Phillips, B., "Simon Fraser University Computer Produced Map Catalogue", Journal of Library Automation, Vol. 2, No. 3 (September 1969), pp. 105-115.

22. Castner, W. G., et al., "The MECCA System- A Modified List Processing Application for Library Collections", Proceedings of the 1966 ACM National Conference, Academic Press, London, 1967, pp. 489-498.

23. Franke, R. D., "Computerized Library Catalog", Datamation, Vol. 14, No. 2 (February 1968), pp. 45-52.

24. Kozlow, R. D., Report on a Library Automation Project Conducted on the Chicago Campus of the University of Illinois, University of Illinois, 1966.

25. Crismond, L. F. and Fatzer, S. B., "Automated Serials Check-In and Binding Procedures at the San Francisco Public Library", Proceedings of the American Society for Information Science, Vol. 6, Greenwood Publishing, New York, pp. 13-20.

26. de Varennes, R., "Computerized Serials Record at Laval University: A Progress Report", Canadian Library, Vol. 24, No. 2 (September 1967), pp. 122-123.

27. Southern, W. A., "Library Applications of Data Processing Equipment at Abbott Laboratories", Proceedings of the 1965 Clinic on Library Applications of Data Processing, Edited by F. B. Jenkins, University of Illinois, 1966, pp. 156-174.

28. Stewart, B. W., "A Computerized Serials Record System for the Texas A&M University Library", Texas A&M University, 1966.

29. Kanasy, J. E., "Circulation Control Systems", Automation in Libraries, Canadian Association of College and University Libraries, 1967, pp. 76-88.

30. Blair, E. J., "An Automated Circulation System and Master Book File for a Medium-Sized Scientific Library", Proceedings of the American Society for Information Science, Vol. 6, Greenwood Publishing, New York, 1969, pp. 21-28.

31. Woods, R. G., "Use of an ICT 1907 Computer in Southampton University Library, Report No. 3", Program, Vol. 1, No. 2, (April 1968), pp. 30-33.

32. Auld, L., "Automated Book Order and Circulation Control Procuedures at the Oakland University Library", Journal of Library Automation, Vol. 1, No. 2 (June 1968), pp. 93-109.

33. Courtright, B., "Automated Circulation Systems", Practical Problems of Library Automation, Documentation Group, Washington, D. C. Chapter, Special Libraries Association, Chicago, 1968, pp. 39-45.

34. Thomson, J. W., and Muller, R. N., "The Computer-Based Book Order System at the University of Michigan Library: A Review and Evaluation", Proceedings of the 1968 Clinic on Library Applications of Data Processing, Edited by D. E. Carroll, University of Illinois, 1969, pp. 54-78.

35. Morris, N. C., "Computer Based Acquisitions System at Texas A&I Iniversity", Journal of Library Automation, Vol. 1, No. 1 (March 1968), pp. 1-12.

36. Teare, R. F., "Experience to Date in Automated Acquisitions at Honnold Library", Proceedings of the American Society for Information Science, Vol. 6, Greenwood Publishing, New York, 1969, pp. 29-37.

37. Grose, M. W. and Jones, B., "The Newcastle University Library Order System", Organization and Handling of Bibliographic Records by Computer, Edited by N. S. M. Cox and M. W. Grose, Archon Books, Hamden, 1967, pp. 158-167.

38. Dobb, T. C., "Simon Fraser's Automated Acquisition System: April 1st, 1966 to April 1st, 1967", Automation in Libraries, Canadian Association of College and University Libraries, 1967, pp. 25-36.

39. Bateman, B. B. and Farris, E. H., "Operating a Multi-library System Using Long-Distance Communications to an On-Line Computer", Proceedings of the American Society for Information Science, Vol. 5, Greenwood Publishing, New York, 1968, pp. 155-162.

40. Kilgour, F. G., "Library Automation", Annual Review of Information Science and Technology, Vol. 4, Edited by C. A. Cuadra, Encyclopaedia Britannica, Chicago, 1969, pp. 305-337.

41. de Varennes, R., "On-Line Serials System at Laval University Library", Journal of Library Automation, Vol. 3, No. 2 (June 1970), pp. 128-141.

42. Kimber, R. T., "Studies at the Queen's University of Belfast on Real-Time Computer Control of Book Circulation", Journal of Documentation, Vol. 22, No. 1 (June 1966), pp. 116-122.

43. Kimber, R. T. "Conversational Circulation", Libri, Vol. 17, No. 2, pp. 131-141.

44. Lazorick, G. J. and Herling, J. P., "A Real-Time Library Circulation System Without Pre-Punched Cards", Proceedings of the American Documentation Institute Annual Meeting, Vol. 4, Thompson Book Co., Washington, pp. 202-206.

45. Stanwood, R. H., "Monograph and Serial Circulation Control", Preprint of paper presented at FID Congress, Buenos Aires, September 21-24, 1970.

46. Kimber, R. T., Automation in Libraries, Pergamon Press, London, 1968.

47. Umstead, C. R. and Croxton, F. E., Automated Library Control Systems, American Data Processing Inc., Detroit, 1969.

48. EDP Systems Development Services, "Report of the Survey of Data Processing Feasibility for the Prince George's County Memorial Library System", EDP Systems Development Services, 1965.

49. Kennedy, R. A., "Bell Laboratories' Real-Time Loan System (BELLREL)", Journal of Library Automation, Vol. 1, No. 2 (June 1968), pp. 128-146.

50. Surace, C. J., "Library Circulation Systems: An Overview", Rand Corporation, Santa Monica, 1970.

51. Hamilton, R. E., "The Illinois State Library 'On-Line' Circulation Control System", Proceedings of the 1968 Clinic on Library Applications of Data Processing, Edited by D. E. Carroll, University of Illinois, 1969, pp. 11-28.

52. Kimber, R. T., "An Operational Computerized Circulation System with On-Line Interrogation Capability", Program, Vol. 2, No. 3 (October 1968), pp. 75-80.

53. Cowell, D., Personal Communication Relating to the On-Line Circulation System at Simon Fraser University, June 3, 1970.

54. Boyd, A. H. and Walden, E. J., "A Simplified On-Line Circulation System", Program, Vol. 3, No. 2 (July 1969), pp. 47-65.

55. Heiliger, E., "Florida Atlantic University Library", Proceedings of the 1965 Clinic on Library Applications of Data Processing, Edited by F. B. Jenkins, University of Illinois, 1966, pp. 92-111.

56. Hayes International Corporation, "Automated Literature Processing Handling and Analysis System - First Generation", Hayes International Corporation, Huntsville, 1967.

57. Redstone Scientific Information Center, "Automation in Libraries", ATLIS Report No. 13, 1966.

58. Boylan, M. N. et al., "Automated Acquisition, Cataloging, and Circulation in a Large Research Library", Lawrence Radiation Laboratory, University of California, Livermore, 1968.

59. Paxton, E. A., Bodie, E. K., and Jacob, M. E., "Integrating Major Library Functions into One Computer-Oriented System", Proceedings of the American Society for Information Science, Vol. 5, Greenwood Publishing, New York, 1968, pp. 141-149.

60. International Business Machines, Mechanized Library Procedures for the IBM Advanced Systems Development Division Library Los Gatos, California, International Business Machines Corp., White Plains, New York, 1967.

61. Scott, J. W., "An Integrated Computer Based Technical Information Processing System in a Small College Library", Journal of Library Automation, Vol. 1, No. 3 (September 1968) pp. 149-158.

62. Kountz, J. C., and Norton, R. E., "BIBLIOS - A Modular System for Library Automation", Datamation, Vol. 16, No. 2 (February 1970), pp. 79-83.

63. Kountz, J. E., and Norton, R. E., "BIBLIOS - A Modular Approach to Total Library ADP", Proceedings of the American Society for Information Science, Vol. 6, Greenwood Publishing, New York, 1969, pp. 39-50.

64. Parker, R., "A Feasibility Study for a Joint Computer Center for Five Washington, D.C. University Libraries", Consortium of Universities of Metropolitan Washington, D.C., Washington, May 1968.

65. Kilgour, F. G., "A Regional Network - Ohio College Library Center", Datamation, Vol. 16, No. 2 (February 1970), pp. 87-89.

66. Kilgour, F. G., "Initial System Design for the Ohio College Center: A Case History", Proceedings of the 1968 Clinic on Library Applications of Data Processing, Edited by D. E. Carroll, University of Illinois, 1969, pp. 79-88.

67. Nugent, W. R., "NELINET - The New England Library Information Network", Proceedings of IFIP Congress 1968, Vol. 2, North-Holland Publishing Co., Amsterdam, 1969, pp. 1315-1320.

68. Hammer, D. P., "Automated Operations in a University Library - A Summary", College & Research Libraries, Vol. 26, No. 1 (January 1965), pp. 19-29.

69. Grosch, A. N., "Implications of On-Line Systems Techniques for a Decentralized Research Library System", College & Research Libraries, Vol. 30, No. 2 (March 1969), pp. 112-118.

70. Warheit, I. A., "Current Developments in Library Mechanization", Special Libraries, Vol. 58, No. 6 (July-August 1967), pp. 420-426.

71. Chicago University, "Development of an Integrated, Computer-Based Bibliographical Data System for a Large University Library", Chicago University, 1968.

72. Fussler, H. H. and Payne, C. T., "Development of an Integrated, Computer-Based Bibliographical Data System for a Large University Library", Chicago University, 1967.

73. Burgess, T., et al., "LOLA - Library On-Line Acquisitions Subsystem", Washington State University, 1968.

74. Swanson, D. R., "Design Requirements for a Future Library", Libraries and Automation, Edited by B. E. Markuson, Library of Congress, Washington, 1964, pp. 11-17.

75. Kilgour, F. G., "Concept of an On-Line Computerized Catalog", Journal of Library Automation, Vol. 3, No. 1 (March 1970), pp. 1-11.

76. Kilgour, F. G., "Effect of Computerization on Acquisitons", Program, Vol. 3, No. 3&4 (November 1969), pp. 95-103.

77. Veaner, A. B., "Bibliographic Automation of Large Library Operations Using a Time-Sharing System (Project BALLOTS)", Research Proposal, Stanford University, 1967.

78. Cunningham, J. L., Schieber, W. D., and Schoffner, R. M., "A Study of the Organization and Search of Bibliographic Holdings Records in On-Line Computer Systems: Phase I", Institute of Library Research, University of California, Berkeley, 1969.

79. Divett, R. T., "Design of a File Structure for a Total System Computer Program for Medical Libraries and Programming of the Book Citation Module", Doctoral Dissertation, University of Utah, 1968.

80. Kessler, M. M., "The MIT Technical Information Project-System Description", Massachusetts Institute of Technology, 1964.

81. Kessler, M. M., "The MIT Technical Information Project", Physics Today, Vol. 18, No. 3 (March 1965) pp. 28-36.

82. Kessler, M. M., et al., "The MIT Technical Information Project - A Prototype System", Proceedings of the ADI Annual Meeting, Vol. 2, Cleaver-Hume Press, London, 1965, pp. 263-268.

83. Mathews, W. D., "The TIP Retrieval System at MIT", Information Retrieval, Edited by G. Schecter, Thompson Book Co., Washington, 1967, pp. 95-108.

84. Burnaugh, H. P., "The BOLD (Bibliographic On-Line Display) System", Information Retrieval, Edited by G. Schecter, Thompson Book Co., Washington, 1967, pp. 53-66.

85. Drew, D. L., et al., "An On-Line Technical Library Reference Retrieval System", American Documentation, Vol. 17, No. 1 (January 1966), pp. 3-7.

86. Summit, R. K., "DIALOG: An Operational On-Line Reference Retrieval System", Proceedings of the 1967 ACM National Conference, Thompson Book Co., Washington, 1967, pp. 51-56.

87. Overhage, C. J. F. and Harman, R. J., INTREX, The M.I.T. Press, Cambridge, 1965.

88. Overhage, C. J. F. and Reintjes, J. F., "Information Transfer Experiments at M.I.T.", Proceedings of IFIP Congress 1968, Vol. 2, North-Holland Publishing Co., Amsterdam, 1969, pp. 1315-1320.

89. Benenfeld, A. R., "Generation and Encoding of the Project INTREX Augmented Catalog Data Base", Massachusetts Institute of Technology, 1968.

90. Massachusetts Institute of Technology, "Project INTREX - Semi-Annual Activity Report, 15 March 1968 to 15 September 1968", Massachusetts Institute of Technology, 1968.

91. Massachusetts Institute of Technology, "Project INTREX - Semi-Annual Activity Report, 15 September 1968 to 15 March 1969", Massachusetts Institute of Technology, 1969.

92. Massachusetts Institute of Technology, "Project INTREX - Semi-Annual Activity Report, 15 March 1969 to 15 September 1969", Massachusetts Institute of Technology, 1969.

93. Reintjes, J. R., "System Characteristics of INTREX", Proceedings of 1969 Spring Joint Computer Conference, AFIPS Press, Montvale, 1969, pp. 457-459.

94. Marcus, R. S., Kugel, P., and Kusik, R. L., "An Experimental Computer-Stored, Augmented Catalog of Professional Literature", Proceedings of 1969 Spring Joint Computer Conference, AFIPS Press, Montvale, 1969, pp. 461-473.

95. Heaps, D. M., et al., "Automation of a UDC Based Library for Searching Purposes", Preprint of a paper presented at 1970 FID Congress, 2nd UDC Mechanization Seminar, Frankfurt, 1-5 June 1970.

96. Atherton, P. and Miller, K. B., "LC/MARC on MOLDS: An Experiment in Computer-Based, Interactive Bibliographic Storage, Search, Retrieval, and Processing", Journal of Library Automation, Vol. 3, No. 2 (June 1970), pp. 142-165.

97. Heaps, D. M. and Sorenson, P., "An On-Line Personal Documentation System", Proceedings of the American Society for Information Science, Vol. 5, Greenwood Publishing, New York, 1968, pp. 201-207.

98. Freeman, R. R. and Atherton, P. "AUDACIOUS - An Experiment with an On-Line, Interactive Reference Retrieval System Using the Universal Decimal Classification as the Index Language in the Field of Nuclear Science", Proceedings of the American Society for Information Science, Vol. 5, Greenwood Publishing, New York, 1968.

99. Green, J. S., "Studies in the Man-System Interface in Libraries", Lehigh University, 1967.

100. Halpin, R., "An On-Line Information Retrieval System with Application to Western Canadian History", Master's Thesis, Department of Computing Science, University of Alberta, 1969.

101. Hillman, D. J. and Kasarda, A. J., "The LEADER Retrieval System", Proceedings of 1969 Spring Joint Computer Conference, AFIPS Press, Montvale, 1969, pp. 447-455.

102. Lefkovitz, D. and Powers, R. V., "A List-Structured Chemical Information Retrieval System", Information Retrieval, Edited by G. Schecter, Thompson Book Co., Washington, 1967, pp. 109-129.

103. Weinberg, P. R., "A Time-Sharing Chemical Information Retrieval System", Doctoral Dissertation, University of Pennsylvania, 1969.

104. Carroll, D. E., et al., "An Interactive Document Retrieval System", Illinois University, Urbana, 1968.

105. Sheldon, R. C., "Design of Information Storage and Retrieval System. Part 3. Development of a Time-Shared Storage and Retrieval System", Massachusetts Institute of Technology, 1967.

106. Cautin, H., et al., "An Experimental On-Line Information Retrieval System", Pennsylvania University, Philadelphia, 1967.

107. Rubinoff, M. et al., "EASY English, A Language for Information Retrieval Through a Remote Typewriter Console", University of Pennsylvania, 1967.

108. Rubinoff, M. et al., "Experimental Evaluation of Information Retrieval Through a Teletypewriter", Communications of the ACM, Vol. 11, No. 9 (September 1968), pp. 598-604.

109. Salton, G., Automatic Information Organization and Retrieval, McGraw-Hill Book Co., New York, 1968, pp. 422-484.

110. International Business Machines, Introduction to IBM System/360 Direct Access Storage Devices and Organization Methods, International Business Machines Corp., White Plains, New York, 1969.

111. Avram, H. D., et al., "MARC Program Research and Development: A Progress Report", Journal of Library Automation, Vol. 2, No. 4 (December 1969), pp. 242-265.

112. Heaps, H. S. and Thiel, L. H., "Optimum Procedures for Economic Information Retrieval", Information Storage and Retrieval, Vol. 6, No. 2 (June 1970), pp. 137-153.

113. Heaps, H. S. and Thiel, L. H., "Program Design for Retrospective Searches on Large Data Bases", Preprint of paper presented at the Third Cranfield International Conference on Mechanized Information Storage and Retrieval Systems, July 1970.

114. Morris, R., "Scatter Storage Techniques", Communications of the ACM, Vol. 11, No. 1 (January 1968), pp. 38-44.

115. Murray, D. M., "A Scatter Storage Scheme for Dictionary Lookups", Journal of Library Automation, Vol. 3, No. 3 (September 1970), pp. 173-201.

116. Buchholz, W., "File Organization and Addressing", IBM Systems Journal, Vol. 2, No. 2 (June 1963), pp. 86-111.

117. Schay, G. and Spruth, W. G., "Analysis of a File Addressing Method", Communications of the ACM, Vol. 5, No. 8 (August 1962), pp. 459-462.

118. Tainter, M., "Addressing for Random-Access Storage with Multiple Bucket Capacities", Journal of the ACM, Vol. 10, No. 3 (July 1963), pp. 307-315.

119. Heising, W. P., "Note on Random Addressing Techniques", IBM Systems Journal, Vol. 2, No. 2 (June 1963), pp. 112-116.

120. Beyer, W. H., Handbook of Tables for Probability and Statistics, The Chemical Rubber Company, Cleveland, 1966.

121. Arkin, H. and Colton, R. R., Tables for Statisticians, Barnes & Noble, New York, 1963.

122. Maurer, W. D., "An Improved Hash Code for Scatter Storage", Communications of the ACM, Vol. 11, No. 1 (January 1968), pp. 35-38.

123. Toyoda, J., Tezuka, Y., and Kasahara, Y., "Analysis of the Address Assignment Problem for Clustered Keys", Journal of the ACM, Vol. 13, No. 4 (October 1966), pp. 526-532.

124. Hopgood, F. R. A., "A Solution to the Table Overflow Problem for Hash Tables", The Computer Bulletin, Vol. 11, No. 4 (March 1968), pp. 297-300.

125. Radke, C. E., "The Use of Quadratic Residue Research", Communications of the ACM, Vol. 13, No. 2 (February 1970), pp. 103-105.

126. Lamport, L., "Comment on Bell's Quadratic Quotient Method for Hash Code Searching", Communications of the ACM, Vol. 13, No. 9 (September 1970), pp. 573-574.

127. Bell, J. R., "The Quadratic Quotient Method: A Hash Code Eliminating Secondary Clustering", Communications of the ACM, Vol. 13, No. 2 (February 1970), pp. 107-109.

128. Reid, W. D., Personal Communication Relating to his Thesis Research, March 1971.

129. Tagliacozzo, R., Rosenberg, L., and Kochen, M., "Access and Recognition: From User's Data to Catalogue Entries", Journal of Documentation, Vol. 26, No. 3 (September 1970), pp. 230-249.

130. Hellerman, H., Digital Computer System Principles, McGraw-Hill Book Co., New York, 1967, pp. 317.

131. Kucera, H. and Francis, W. N., Computational Analysis of Present-Day American English, Brown University Press, Providence, 1967.

132. Bird, E., Personal Communication Relating to Statistical Studies Done by the Systems Planning and Development Department of the Cameron Library of the University of Alberta, March 1971.

133. Prywes, N. S. and Litofsky, B., "All Automatic Processing for a Large Library", Proceedings of 1970 Spring Joint Computer Conference, AFIPS Press, Montvale, 1970, pp. 323-331.

134. University of Alberta Computing Center, "Files and Devices", Computing Center Publication, University of Alberta, 1970.

135. Warheit, I. A., "File Organization of Library Records", Journal of Library Automation, Vol. 2, No. 1 (March 1969), pp. 20-30.

136. Henderson, J. W. and Rosenthal, J. A. (eds.), Library Catalogs: Their Preservation and Maintenance by Photographic and Automated Techniques, M.I.T. Report No. 14, The M.I.T. Press, Cambridge, 1968.

137. Curtice, R. M., "Experimental Retrieval Systems Studies. Report No. 1 Magnetic Tape and Disc File Organization for Retrieval", Master's Thesis, Lehigh University, 1966.

138. Dodd, G. G., "Elements of Data Management Systems", Computing Surveys, Vol. 1, No. 2 (June 1969), pp. 117-133.

APPENDICES

APPENDIX I

ALPHABETICAL INDEX TO THE ON-LINE COMMANDS OF THE IT SYSTEM

ALTER 64
AVAILABLE 62
BRING 59
DAY 65
DEMAND 62
ENROLL 63
ENQUIRE 59
FIND 61
LIST 61
LOAN 60
LOOKUP 64
REMOVE 65
RENEW 60
REQUEST 59
RESERVE 58
RETURN 60
SEARCH 55
STOP 66
TRACE 66

APPENDIX II

SAMPLE TERMINAL SESSION

Alphabetical Index to the Commands That Are Demonstrated

ALTER 309

AVAILABLE 306

ENROLL 307

ENQUIRE 301

FIND 303, 308

LIST 305

LOAN 302, 304

LOOKUP 305

REMOVE 304

RENEW 301

REQUEST 301, 302, 303. 309

RETURN 303

SEARCH 297, 298, 299, 300

TRACE 310

```
 source library

#$SET ECHO=OFF
# 19:20.33
# 19:20.34 .089 RC=0
# FILE "OFFLINE.I/O" HAS BEEN CREATED.
# 19:20.45




    LIBRARY AUTOMATION...VERSION 3.2...JUNE 1, 1970.
  C) APR 29, 1971
       1427

  C) ENTER A COMMAND
 search cox
 librar computer

  C) COX, NIGEL S. M.
        THE COMPUTER AND THE LIBRARY,  THE ROLE OF THE COMPUTER IN THE
        ORGANIZATION AND HANDLING OF INFORMATION IN LIBRARIES,  BY
        N. S. M. COX, J. D. DEWS, AND J. L. DOLBY.
        (UNIVERSITY LIBRARY PUBLICATIONS, NEWCASTLE UPON TYNE, 4)
          UNIVERSITY OF NEWCASTLE UPON TYNE LIBRARY, 1966, 95 P, 24 CM C.4
          CATALOG NO. Z678.9C87          IN COMPUTER SCIENCE LIBRARY
          STATUS:  BORROWED

  C) ENTER A COMMAND
```

```
search cox
librar|statistic

 C) COX, DAVID ROXBEE
       THE STATISTICAL ANALYSIS OF SERIES OF EVENTS,  BY D.R.COX AND
       P.A.W.LEWIS
       (METHUEN'S MONOGRAPHS ON APPLIED PROBABILITY AND STATISTICS)
         LONDON, METHUEN, 1966, 285 P. ILLUS. 22 CM. C.5
         CATALOG NO. QA276C87          IN COMPUTER SCIENCE LIBRARY
         STATUS:  AVAILABLE

 C) COX, NIGEL S. M.
       THE COMPUTER AND THE LIBRARY,  THE ROLE OF THE COMPUTER IN THE
       ORGANIZATION AND HANDLING OF INFORMATION IN LIBRARIES,  BY
       N. S. M. COX, J. D. DEWS, AND J. L. DOLBY.
       (UNIVERSITY LIBRARY PUBLICATIONS, NEWCASTLE UPON TYNE, 4)
         UNIVERSITY OF NEWCASTLE UPON TYNE LIBRARY, 1966, 95 P, 24 CM C.4
         CATALOG NO. Z678.9C87         IN COMPUTER SCIENCE LIBRARY
         STATUS:  BORROWED

 C) ENTER A COMMAND
search cox  nsm
*

 C) COX, NIGEL S. M.
       THE COMPUTER AND THE LIBRARY,  THE ROLE OF THE COMPUTER IN THE
       ORGANIZATION AND HANDLING OF INFORMATION IN LIBRARIES,  BY
       N. S. M. COX, J. D. DEWS, AND J. L. DOLBY.
       (UNIVERSITY LIBRARY PUBLICATIONS, NEWCASTLE UPON TYNE, 4)
         UNIVERSITY OF NEWCASTLE UPON TYNE LIBRARY, 1966, 95 P, 24 CM C.4
         CATALOG NO. Z678.9C87         IN COMPUTER SCIENCE LIBRARY
         STATUS:  BORROWED

 C) ENTER A COMMAND
```

```
 C) ENTER A COMMAND
search *
librar computer|automat|system

 C) COX, NIGEL S. M.
       THE COMPUTER AND THE LIBRARY,  THE ROLE OF THE COMPUTER IN THE
       ORGANIZATION AND HANDLING OF INFORMATION IN LIBRARIES,  BY
       N. S. M. COX, J. D. DEWS, AND J. L. DOLBY.
       (UNIVERSITY LIBRARY PUBLICATIONS, NEWCASTLE UPON TYNE, 4)
         UNIVERSITY OF NEWCASTLE UPON TYNE LIBRARY, 1966, 95 P, 24 CM C.4
         CATALOG NO. Z678.9C87          IN COMPUTER SCIENCE LIBRARY
         STATUS:  BORROWED

 C) KENT, ALLEN, EDITOR.
       LIBRARY PLANNING FOR AUTOMATION.  WORKING PAPERS AND PROCEEDINGS
       OF A CONFERENCE HELD AT THE UNIVERSITY OF PITTSBURGH,
       JUNE 2 TO 3, 1964. (THE KNOWLEDGE AVAILABILITY SYSTEMS SERIES)
         WASHINGTON, SPARTAN BOOKS, 1965, 193 P., ILLUS., 24 CM.
         CATALOG NO. Z678.9K37          IN COMPUTER SCIENCE LIBRARY
         STATUS:  AVAILABLE

 C) MARKUSON, BARBARA EVANS, EDITOR.
       LIBRARIES AND AUTOMATION;  CONFERENCE PROCEEDINGS, HELD MAY 26-30,
       1963, UNDER SPONSORSHIP OF THE LIBRARY OF CONGRESS, NATIONAL
       SCIENCE FOUNDATION, COUNCIL ON LIBRARY RESOURSES, INC.
         WASHINGTON, LIBRARY OF CONGRESS, 1964, 268 P., ILLUS, 27 CM, C.2
         CATALOG NO. Z699C74-1963       IN COMPUTER SCIENCE LIBRARY
         STATUS:  BORROWED

 C) MORSE, PHILIP MCCORD, 1903-
       LIBRARY EFFECTIVENESS: A SYSTEMS APPROACH
         CAMBRIDGE, MASS., M.I.T. PRESS, 1968, 207 P. 24 CM. C.3
         CATALOG NO. Z670.5M88          IN COMPUTER SCIENCE LIBRARY
         STATUS:  BORROWED

 C) ENTER A COMMAND
```

```
search *
librar comput|automat|system ~plan

 C) COX, NIGEL S. M.
       THE COMPUTER AND THE LIBRARY,  THE ROLE OF THE COMPUTER IN THE
       ORGANIZATION AND HANDLING OF INFORMATION IN LIBRARIES,  BY
       N. S. M. COX, J. D. DEWS, AND J. L. DOLBY.
       (UNIVERSITY LIBRARY PUBLICATIONS, NEWCASTLE UPON TYNE, 4)
         UNIVERSITY OF NEWCASTLE UPON TYNE LIBRARY, 1966, 95 P, 24 CM C.4
         CATALOG NO. Z678.9C87          IN COMPUTER SCIENCE LIBRARY
         STATUS:  BORROWED

 C) MARKUSON, BARBARA EVANS, EDITOR.
       LIBRARIES AND AUTOMATION;  CONFERENCE PROCEEDINGS, HELD MAY 26-30,
       1963, UNDER SPONSORSHIP OF THE LIBRARY OF CONGRESS, NATIONAL
       SCIENCE FOUNDATION, COUNCIL ON LIBRARY RESOURSES, INC.
         WASHINGTON, LIBRARY OF CONGRESS, 1964, 268 P., ILLUS, 27 CM, C.2
         CATALOG NO. Z699C74-1963       IN COMPUTER SCIENCE LIBRARY
         STATUS:  BORROWED

 C) MORSE, PHILIP MCCORD, 1903-
       LIBRARY EFFECTIVENESS: A SYSTEMS APPROACH
         CAMBRIDGE, MASS., M.I.T. PRESS, 1968, 207 P. 24 CM. C.3
         CATALOG NO. Z670.5M88          IN COMPUTER SCIENCE LIBRARY
         STATUS:  BORROWED

 C) ENTER A COMMAND
```

```
request morse, p.m.
z670.5m88
dimsdale, jjf
C) YOUR RESERVATION IS NO. 1 ON COPY NO.  1 WITH -38 DAYS REMAINING ON ITS CURRENT LOAN.
 C) EACH DAY TYPE:
    ENQUIRE MORSE                PM
    Z670.5M88
        68
 C) TRANSACTION COMPLETED

 C) ENTER A COMMAND

enquire morse  pm
z670.5m88
dimsdale, j.j.f.
 C) NOT YET AVAILABLE

 C) ENTER A COMMAND

renew  morse  pm
z670.5m88
14

1427

 C) CANNOT RENEW THIS BOOK AS IT HAS BEEN RESERVED AND YOUR TIME IS UP.


C) LIBRARIAN:  HOLD THIS BOOK FOR USER NO.  68


 C) TRANSACTION COMPLETED

 C) ENTER A COMMAND
 C) INVALID COMMAND..
```

```
?

 C) ENTER A COMMAND
loan  morse  pm
?
 C) ENTER CATALOG NUMBER
z670.5m88
?
 C) ENTER USER'S IDENTIFICATION.
68

?
  C) ENTER PASSWORD.

1427

 C) TRANSACTION COMPLETED

 C) ENTER A COMMAND
request morse  pm
z670.5m88
14
C) YOUR RESERVATION IS NO. 1 ON COPY NO.  1 WITH  14 DAYS REMAINING ON ITS CURRENT LOAN.
 C) EACH DAY TYPE:
    ENQUIRE MORSE                PM
    Z670.5M88
        14
 C) TRANSACTION COMPLETED

 C) ENTER A COMMAND
```

```
request morse  pm
z670.5m88
heaps, d.m.
 C) THIS RESERVATION CANNOT BE ACCEPTED AS YOU ALREADY HAVE THIS BOOK ON LOAN OR ON RESERVE.

 C) ENTER A COMMAND
find morse  pm
z670.5m88

1427


AUTHOR:  MORSE                PM          CATALOG NUMBER:  Z670.5M88

     COPY NO.  1     USER:     68  DIMSDALE        JJF      DATE BORROWED:  APR 29, 1971

                  RESERVATION NO.  1 FOR USER:     14  HEAPS           DM

C) DELETE A RESERVATION?
no

 C) ENTER A COMMAND
return morse  pm
z670.5m88
68

1427


C) LIBRARIAN:  HOLD THIS BOOK FOR USER NO.  14

 C) TRANSACTION COMPLETED

 C) ENTER A COMMAND
```

```
 C) ENTER A COMMAND
remove
235

1427

SURNAME AND INITIALS STATUS TITLE ADDRESS                    PHONE  DEPT. IDNUMBER LIBID DATE ENROLLED
FAMIGLIETTI     LG        3     2                                    345667009   235 APR 29, 1971
C) IS THIS THE RECORD THAT YOU WISH TO REMOVE?
yes
FAMIGLIETTI     LG     HAS BEEN REMOVED FROM THE USER DIRECTORY

 C) ENTER A COMMAND
loan cox  dr
qa273c867
famiglietti  lg
FAMIGLIETTI     LG
   IS NOT IN THE USER DIRECTORY

 C) ENTER A COMMAND
loan cox  dr
qa273c867
dimsdale  sj

1427

 C) TRANSACTION COMPLETED

 C) ENTER A COMMAND
```

```
list
heaps  dm

1427


 LIST OF BOOKS ON LOAN TO      HEAPS              DM              APR 29, 1971

AUTHOR                          CATALOG NUMBER      OVERDUE      REQUESTED

HOUGHTON              B         Z699H83             XXXX         XXXX

MINSKY                ML        QA267M66            XXXX         XXXX

 C) ENTER A COMMAND
lookup
heaps  dm

1427

SURNAME AND INITIALS STATUS TITLE ADDRESS                    PHONE  DEPT. IDNUMBER LIBID DATE ENROLLED
HEAPS           DM        1     5 601 GSB                    3977     CS 616921448    14 AUG 15, 1969
DO YOU WISH TO MAKE ANY CHANGES?
no

 C) ENTER A COMMAND
```

```
 C) ENTER A COMMAND
enroll

1427


 C) TYPE IN THE FOLLOWING INFORMATION ABOUT EACH NEW USER.

    EACH RESPONSE MUST BEGIN DIRECTLY UNDER THE FIRST LETTER OF THE REQUIRED ITEM.

SURNAME         INIT STATUS ADDRESS                 PHONE   DEPT        IDNUMBER
dimsdale        sj          #104-8923-79ave.        4338846 sc          712363
DIMSDALE        SJ   0 0    #104-8923-79AVE.        4338846 SC             712363  233
cotterman       ww   1 3    #12 willow cresent        3478  pp        909808312
COTTERMAN       WW   1 3    #12 WILLOW CRESENT        3478  PP           9808312  234
famiglietti     lg   3 xx    new york                                  345t67009
 C) INVALID CHARACTER..X  !!!
 C) ** TITLE **
 C) INVALID CHARACTER..T  !!!
 C) ** IDNUMBER **


*** THE FOLLOWING ENTRY HAS BEEN REJECTED BECAUSE OF ERRORS NOTED ABOVE. ***

FAMIGLIETTI     LG   3 XX    NEW YORK                                  345T67009
famiglietti     lg   3  2                                            345667009
FAMIGLIETTI     LG    3 2                                            345667009  235
*

 C) ENTER A COMMAND
```

```
 C) ENTER A COMMAND
find minsky  ml
qa267m66

1427

AUTHOR:  MINSKY                ML          CATALOG NUMBER:  QA267M66

     COPY NO.  1     USER:      14  HEAPS              DM        DATE BORROWED:  MAR 26, 1971

                     RESERVATION NO.  1 FOR USER:      24  HEYWORTH          J

                     RESERVATION NO.  2 FOR USER:     234  COTTERMAN         WJ

C) DELETE A RESERVATION?
no

     COPY NO.  2     IN LIBRARY

                     RESERVATION NO.  1 FOR USER:      22  VON_MAYDELL       U

C) DELETE A RESERVATION?
no

 C) ENTER A COMMAND
```

```
 C) ENTER A COMMAND
alter
9808312

1427

TO MAKE A CHANGE ENTER THE NEW INFORMATION DIRECTLY UNDERNEATH THE OLD.

SURNAME    AND    INIT STATUS ADDRESS                    PHONE   DEPT        IDNUMBER          LIBRARY ID

COTTERMAN          WW   1 3    #12 WILLOW CRESENT        3478   PP          9808312           234
                   wj                                          ***         909808312
 C) IS THIS CORRECT?

COTTERMAN          WJ   1 3    #12 WILLOW CRESENT        3478               9098083
no
TO MAKE A CHANGE ENTER THE NEW INFORMATION DIRECTLY UNDERNEATH THE OLD.

SURNAME    AND    INIT STATUS ADDRESS                    PHONE   DEPT        IDNUMBER          LIBRARY ID

COTTERMAN          WJ   1 3    #12 WILLOW CRESENT        3478               9098083           234
                                                                            909808312
 C) IS THIS CORRECT?

COTTERMAN          WJ   1 3    #12 WILLOW CRESENT        3478               909808312
yes

 C) ENTER A COMMAND
request minsky  m 1
qa267m66
909808312
C) YOUR RESERVATION IS NO. 2 ON COPY NO.  1 WITH -20 DAYS REMAINING ON ITS CURRENT LOAN.
 C) EACH DAY TYPE:
    ENQUIRE MINSKY              ML
    QA267M66
       234
 C) TRANSACTION COMPLETED

 C) ENTER A COMMAND
```

```
 C) ENTER A COMMAND
trace
1427

search cox  dr
COX                    DR
  362    HARTREE                DR
  181    DALE                   E
   90    BORKO                  H
  135    CHERRY                 C
  158    CONWAY                 RW
  169    COX                    DR
 ** INDX01=   200
theory renewal|time|series
.AND.
.OR.      6THEORY
.AND.
.OR.      7RENEWAL
.OR.      4TIME
.OR.      6SERIES

 C) COX, DAVID ROXBEE
       RENEWAL THEORY
       (METHUEN'S MONOGRAPHS ON APPLIED PROBABILITY AND STATISTICS)
         LONDON, METHUEN, 1967, 142 P. ILLUS. 19 CM.
         CATALOG NO. HA33C87           IN COMPUTER SCIENCE LIBRARY
         STATUS:  BORROWED
                                              200

 C) ENTER A COMMAND
```

APPENDIX III

ALPHABETICAL INDEX TO THE FILES OF THE IT SYSTEM

AUTHORS 82
CATALOG 72
DELETIONS 105
FREELOCS 96
FREESPACE 95
FREESTOR 95
LOANS 80
NEWBOOKS 97
NEWENTRIES 104
NEWUSERS 106
OFFLINE.I/O 114
OVERDUES 94
USER ID NUMBER INDEX 92
USER NAME INDEX 92
USERINDEX 92
USERS 86
SAVEFILE 104

APPENDIX IV

PROGRAMS OF THE IT SYSTEM

Alphabetical Index to FORTRAN IV Programs

ALTER 316

BINS1 321

BINS2 322

BINS3 322

BLKDATA3 323

BLKDAT4 324

BRING 324

CHANGE 325

CHECK 326

DATA 327

DAY 330

DELET 332

DEMAND 334

ENROLL 336

ENROLL:MAIN 342

ENQUIR 343

IDENT 344

INDXR 347

LBOOK 347

LIST 351

MAINLIB 355

MAINUPDATE 361

PICKUP.MAIN 397

QUICK 363

RESERV 364

SAVE 369

SAVE.BATCH 373

SEARCH 373

SETUP 382

SORT1 386

SORT2 387

STATUS 389

TRNSFR 389

UPDATE 390

UPDATE:MAIN 397

Alphabetical Index to Entry Points in the FORTRAN Programs

ALOAN 365

ALTER 316

AVAIL 352

BINS1 321

BINS2 322

BINS3 322

BRING 324

CHANGE 325

CHECK 326

CHECKR 326

CONTIN 391

314

COPY 370

DATA 327

DATE 331

DAY 330

DEMAND 334

DELET 332

DISPLA 371

DUMP 370

EBCDIC 327

ENROLL 336

ENQUIR 343

FINDIT 353

IDENT 344

IDENT2 345

INDXR 347

LBOOK 347

LIST1 351

LIST2 351

LOOKUP 320

LOOK1 383

LOOK2 384

LOOK3 384

PUT 385

PUT2 385

OBOOK 349

QUICK 363

RBOOK 348

REMOVE 320

RESERV 364

RESOR1 340

RESOR2 340

SAVE (batch-mode) 373

SAVE (on-line) 369

SEARCH 373

SETUP 382

SORT1 386

SORT2 387

STATUS 389

TITLE 374

TRNSFR 389

UPDATE 390

Alphabetical Index to the MTS Command Pages

BATCHENROLL 398

BATCHUPDATE 398

LIBRARY 398

PICKUP 399

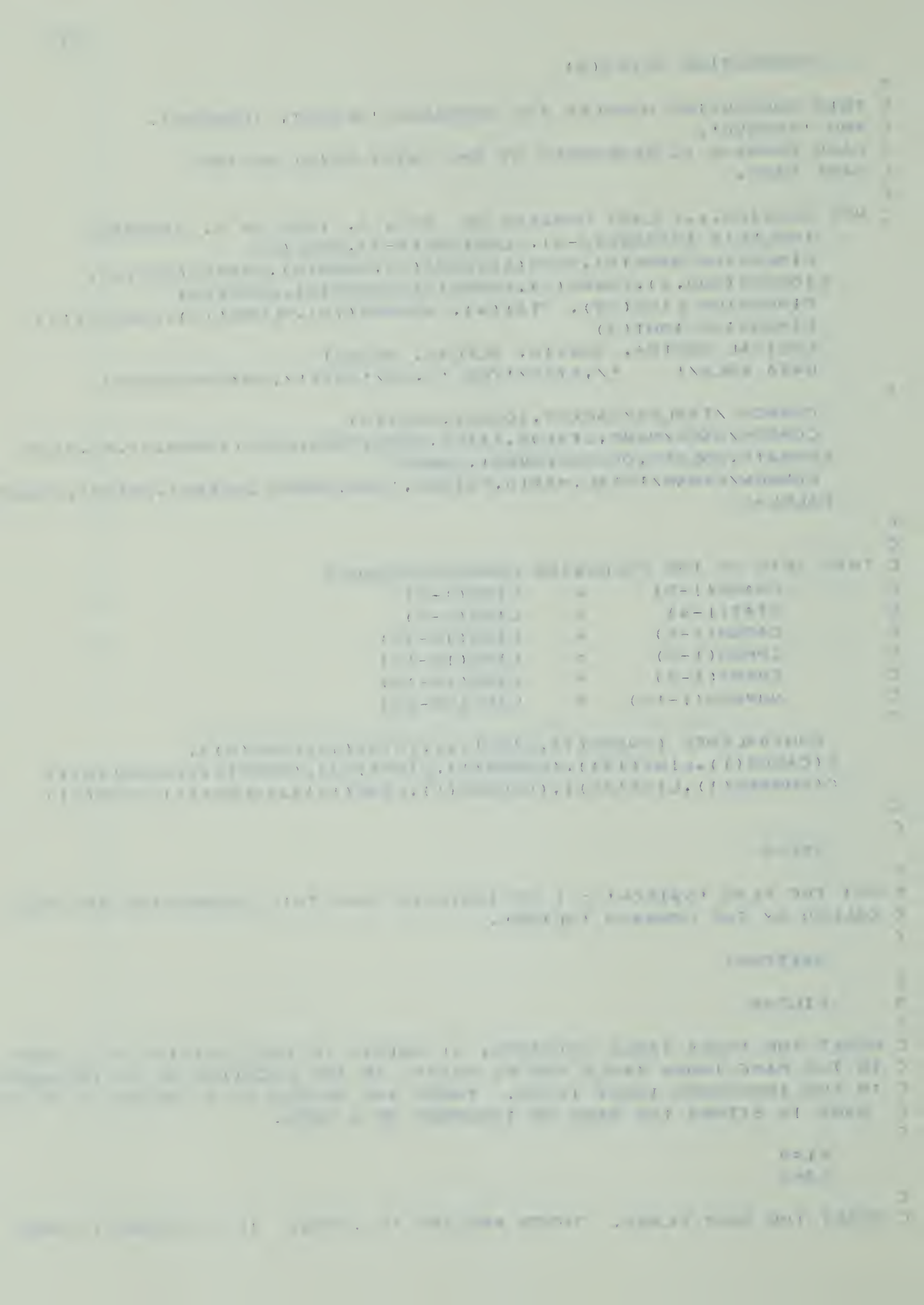
XEQ.ENROLL 399

XEQ.LIBRARY 399

XEQ.PICKUP 400

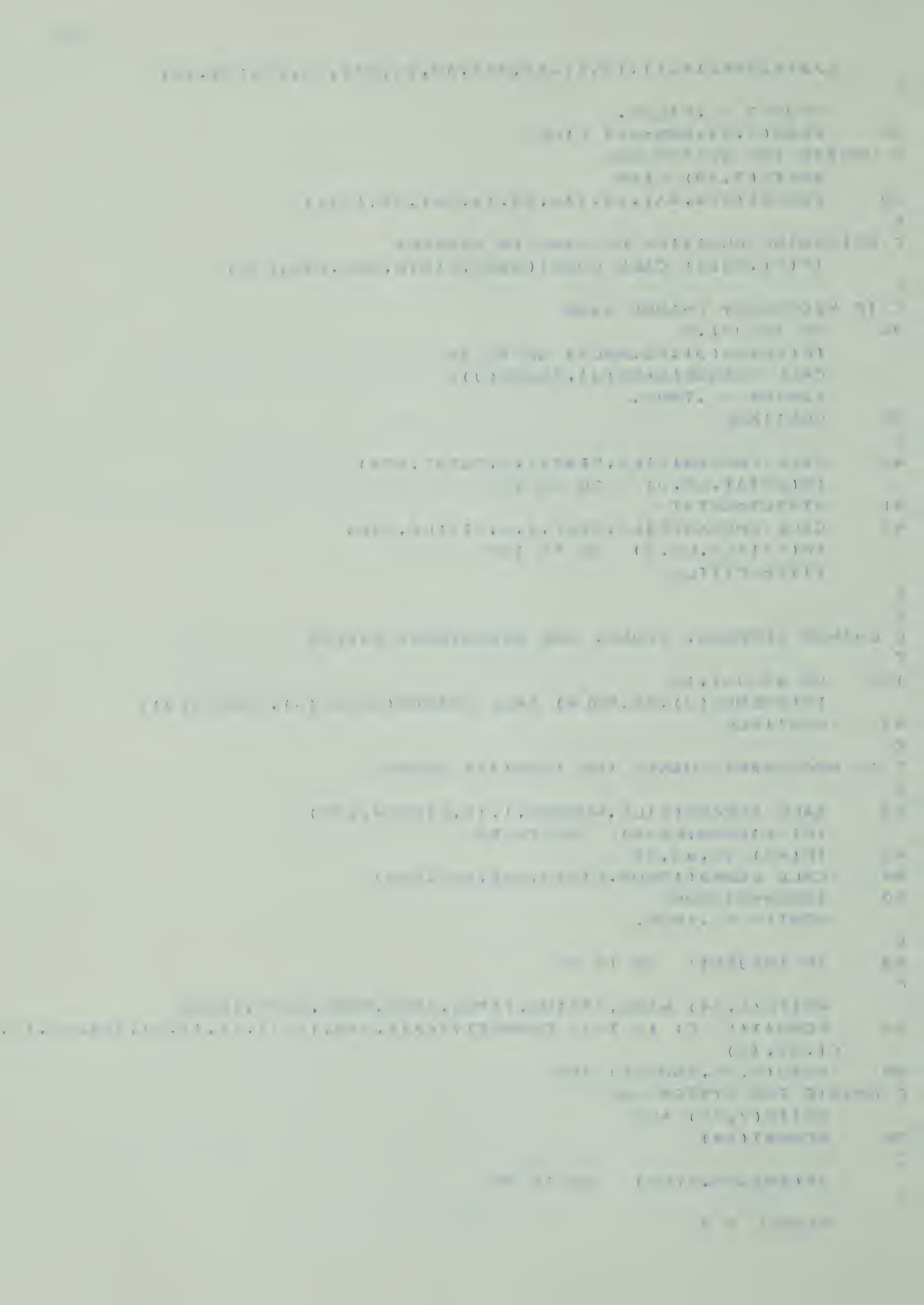
XEQ.UPDATE 400

```
      SUBROUTINE ALTER(*)
C
C THIS SUBROUTINE HANDLES THE COMMANDS 'ALTER', 'LOOKUP',
C AND 'REMOVE'.
C EACH COMMAND IS PERFORMED BY THE ENTRY POINT OF THE
C SAME NAME.
C
C MTS VERSION.... LAST UPDATED ON  OCT. 4, 1970 BY J. DIMSDALE
      IMPLICIT INTEGER(A-W), LOGICAL(X-Z),REAL($)
      DIMENSION NAME(5),ADDR(6),DEPT(2),PHON(2),NAMKEY(500,6),
     CIDNKEY(500,2),CNAME(5),CPHON(2),CADDR(6),CDEPT(2)
      DIMENSION LINE(29), STAT(4), NUMBER(10),BLOCK(10),CBLOCK(10)
      DIMENSION IOUT(3)
      LOGICAL SORTNA, SORTID, ALFLAG, REJECT
      DATA KBLA/'    '/,KYES/'YES '/,KZ/'ZZZZ'/,K9/999999999/
C
      COMMON /TABLES/NAMKEY,IDNKEY,FSL(19)
      COMMON/USER/NAME,STATUS,TITLE,ADDR,PHON,DEPT,IDNUM,K1,K2,ACCT,
     CENDATE,ONLOAN,OVRDUE,OVER1,DUMMY
      COMMON/PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL,USTOR1,OSTOR1,FSIN24,
     CALFLAG
C
C
C THIS SETS UP THE FOLLOWING CORRESPONDENCE:
C         CNAME(1-5)      =    LINE(1-5)
C         STAT(1-4)       =    LINE(6-9)
C         CADDR(1-6)      =    LINE(10-15)
C         CPHON(1-2)      =    LINE(16-17)
C         CDEPT(1-2)      =    LINE(18-19)
C         NUMBER(1-10)    =    LINE(20-29)
C
      EQUIVALENCE (CNAME(1),LINE(1)),(STAT(1),LINE(6)),
     C(CADDR(1),LINE(10)),(CPHON(1),LINE(16)),(CDEPT(1),LINE(18)),
     C(NUMBER(1),LINE(20)),(CBLOCK(1),LINE(10)),(BLOCK(1),ADDR(1))
C
C
      SEE=0
C
C SET THE FLAG 'SWITCH' = 1 TO INDICATE THAT THIS SUBROUTINE HAS BEEN
C CALLED BY THE COMMAND 'ALTER'.
C
      SWITCH=1
C
5     FILE=6
C
C RESET THE INDEX TABLE POINTERS. K1 POINTS TO THE LOCATION OF A NAME
C IN THE NAME INDEX TABLE AND K2 POINTS TO THE LOCATION OF AN IDNUMBER
C IN THE INDNUMBER INDEX TABLE.  THESE ARE NEEDED IF A CHANGE IS TO BE
C  MADE IN EITHER THE NAME OR IDNUMBER OF A USER.
C
      K1=0
      K2=0
C
C RESET THE SORT FLAGS.  THSES ARE SET TO .TRUE.  IF A CHANGE IS MADE
```



```
C IN A USER'S NAME OR IN A USER'S IDNUMBER .  THIS IN TURN CAUSES THE
C INDEX TABLES TO BE RESORTED.
C
      SORTNA = .FALSE.
      SORTID = .FALSE.
C
      EXIT=2
C
C
      IF (SWITCH.EQ.1)  GO TO 29
      CALL DAY(ENDATE,IOUT)
C
C
C    FOLLOWING SECTION TEMPORARILY DELETED WHILE  MODULE LBOOK IS
C NON-OPERATIONAL.    SECTION BETWEEN THE '*' S SUBSTITUTES.
C
C     WRITE(6,10) NAME,STATUS,TITLE,ADDR,PHON,DEPT,IDNUM,DUMMY,ENDATE,
C    CACCT,ONLOAN,OVRDUE
C 10  FORMAT(' SURNAME AND INITIALS STATUS TITLE ADDRESS
C    C PHONE  DEPT. IDNUMBER LIBID DATE ENROLLED'/1X,5A4,I7,I6,1X,7A4,A3
C    C,2X,2A1,I10,I6,I14//' ACCT BOOKS BORROWED BOOKS OVERDUE'/
C    C1X,I4,I14,I13)
C
C
C
C ********
      WRITE(6,10) NAME,STATUS,TITLE,ADDR,PHON,DEPT,IDNUM,DUMMY,IOUT
  10  FORMAT(' SURNAME AND INITIALS STATUS TITLE ADDRESS
     C PHONE  DEPT. IDNUMBER LIBID DATE ENROLLED'/1X,5A4,I7,I6,1X,7A4,A3
     C,2X,2A1,I10,I6, 1X,A4,I2,', 19',I2)
C ********
24    GO TO(29,25,155),SWITCH
C
C
25    WRITE(6,26)
26    FORMAT(' DO YOU WISH TO MAKE ANY CHANGES?')
101   READ(5,56,END=102) ANS
C UPDATE THE SYSTEM LOG
      WRITE(7,56) ANS
      IF(ANS-KYES) 65,29,65
C
102   WRITE(6,67)
      GO TO 101
C
29    SEE=-1
C
C
C THIS SECTION HANDLES ALTERATIONS IN A USER'S RECORD
C
30    WRITE(6,31) NAME,STATUS,TITLE,ADDR,PHON,DEPT,IDNUM,DUMMY
31    FORMAT(' TO MAKE A CHANGE ENTER THE NEW INFORMATION DIRECTLY',
     C' UNDERNEATH THE OLD.'//,
     C' SURNAME   AND   INIT STATUS ADDRESS                    PHONE   DEPT
     C       IDNUMBER            LIBRARY ID NUMBER',
```

```
     C//1X,5A4,1X,I1,1X,I1,4X,7A4,A3,1X,2A1,9X,I9,10X,I4)
C
      REJECT = .FALSE.
32    READ(5,33,END=65) LINE
C UPDATE THE SYSTEM LOG
      WRITE(7,33) LINE
33    FORMAT(5A4,4A1,4X,7A4,A3,1X,2A1,8X,10A1)
C
C DETERMINE POSITION OF NAME IN NAMEKEY
      IF(K1.EQ.0) CALL LOOK1(NAME,LIBID,SEE,&34,&80)
C
C IF NECESSARY CHANGE NAME
34    DO 35 J=1,5
      IF(CNAME(J).EQ.KBLA) GO TO 35
      CALL CHANGE(NAME(J),CNAME(J))
      SORTNA = .TRUE.
35    CONTINUE
C
40    CALL CHECKR(FILE,STAT,1,2,CSTAT,&74)
      IF(CSTAT.LE.0)   GO TO 42
41    STATUS=CSTAT
42    CALL CHECKR(FILE,STAT,3,4,CTITLE,&76)
      IF(CTITLE.LE.0)  GO TO 103
      TITLE=CTITLE
C
C
C CHANGE ADDRESS, PHONE, AND DEPARTMENT FIELDS
C
103   DO 43 J=1,10
      IF(CBLOCK(J).NE.KBLA) CALL CHANGE(BLOCK(J),CBLOCK(J))
43    CONTINUE
C
C IF NECESSARY CHANGE THE IDENTITY NUMBER
C
47    CALL CHECKR(FILE,NUMBER,1,10,CIDNUM,&78)
      IF( CIDNUM.LE.0)   GO TO 53
48    IF(K2) 49,49,50
49    CALL LOOK2(IDNUM,LIBID,SEE,&50,&80)
50    IDNUM=CIDNUM
      SORTID = .TRUE.
C
53    IF (REJECT)  GO TO 70
C
      WRITE(6,54) NAME,STATUS,TITLE,ADDR,PHON,DEPT,IDNUM
54    FORMAT('  C) IS THIS CORRECT?'//1X,5A4,1X,I1,1X,I1,4X,7A4,A3,1X,2A
     C1,9X,I9)
55    READ(5,56,END=66) ANS
C UPDATE THE SYSTEM LOG
      WRITE(7,56) ANS
56    FORMAT(A4)
C
      IF(ANS.NE.KYES)   GO TO 30
C
      NUMDEL = 0
```



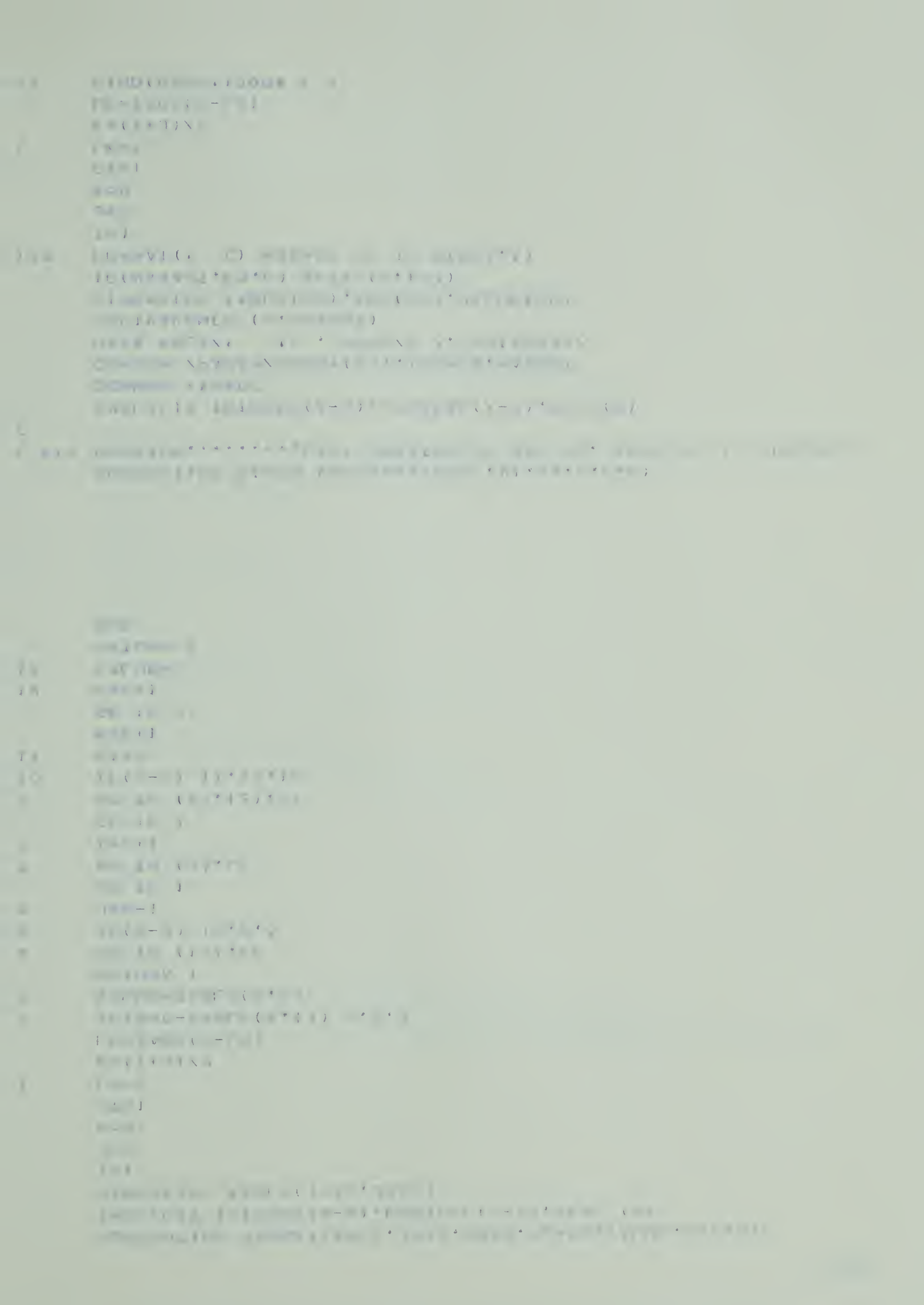
```
      ALFLAG = .TRUE.
C
      IF(.NOT.SORTNA)   GO TO 59
C
      DO 58 J = 1,5
      NAMKEY(K1,J) =  NAME(J)
58    CONTINUE
      CALL RESOR1(&59,&80,EXIT)
C
59    IF(.NOT.SORTID)   GO TO 61
C
      IDNKEY(K2,1)= IDNUM
60    CALL RESOR2(&61,&80,EXIT)
C
61    CALL PUT(&65)
65    RETURN 1
C
C
C
66    WRITE(6,67)
67    FORMAT(' YES OR NO, PLEASE')
      GO TO 55
C
C
C
C MESSAGES FOR ERRORS ENCOUNTERED DURING  PROCESSING OF ALTERATIONS IN
C A USER'S RECORD.
C
70    WRITE(FILE,71) LINE
71    FORMAT(//' *** THE FOLLOWING ENTRY HAS BEEN REJECTED BECAUSE OF ER
     CRORS NOTED ABOVE. ***'//1X,5A4,4A1,4X,7A4,A3,1X,2A1,9X,9A1)
      REJECT = .FALSE.
      GO TO 32
74    WRITE(6,75)
75    FORMAT('  C) ** STATUS **')
      REJECT = .TRUE.
      GO TO 42
76    WRITE(6,77)
77    FORMAT('  C) ** TITLE **')
      REJECT = .TRUE.
      GO TO 103
78    WRITE(6,79)
79    FORMAT('  C) ** IDNUMBER **')
      GO TO 70
C
C
80    WRITE(6,81)
81     FORMAT('  C) DO YOU WISH TO TRY AGAIN?')
      READ(5,56,END=65)   ANS
C UPDATE THE SYSTEM LOG
      WRITE(7,56) ANS
      IF(ANS.NE.KYES)   RETURN 1
      GO TO 30
C
```

```
C
C
      ENTRY LOOKUP(*)
C
C
      SEE=1
91    SWITCH=2
      GO TO 5
C
C
C
      ENTRY REMOVE(*)
C
C
150   SEE=1
      SWITCH=3
      GO TO 5
C
C
155   WRITE(6,156)
156   FORMAT(' C) IS THIS THE RECORD THAT YOU WISH TO REMOVE?')
157   READ(5,56,END=158) ANS
C UPDATE THE SYSTEM LOG
      WRITE(7,56) ANS
      IF(ANS.NE.KYES)  RETURN 1
      IF(ONLOAN.LE.0)   GO TO 159
      WRITE(6,301) ONLOAN
301   FORMAT('  C) THIS USER HAS ',I2,' BOOK(S) ON LOAN AND , THEREFORE,
     C MUST NOT BE REMOVED.')
      RETURN 1
158   WRITE(6,67)
      GO TO 157
159   FSINDX=FSINDX+1
      IF(FSINDX.LE.19) GO TO 300
      XOLD = .FALSE.
      CALL PUT2(XOLD)
      FSINDX=1
300   FSL(FSINDX)=DUMMY
      SEE=-1
      IF(K1) 160,160,165
160   CALL LOOK1(NAME,LIBID,SEE,&165,&250)
165   NAMKEY(K1,1)=KZ
      IF(K2) 170,170,175
170   CALL LOOK2(IDNUM,LIBID,SEE,&175,&250)
175   IDNKEY(K2,1)=K9
      EXIT=1
      NUMDEL=1
      ALFLAG = .FALSE.
      CALL RESOR1(&180,&250,EXIT)
180   WRITE(6,181) NAME
181   FORMAT(1X,5A4,'   HAS BEEN REMOVED FROM THE USER DIRECTORY')
      ACCT=4
      CALL PUT(&250)
      XOLD = .TRUE.
```

```
      CALL PUT2(XOLD)
250   RETURN 1
      END



      SUBROUTINE BINS1(TABLE,IMAX,JMAX,N,WT,ARG,WA,VALUE,K,*,*)
      IMPLICIT INTEGER(A-W),LOGICAL(X-Z),REAL($)
      DIMENSION TABLE(IMAX,JMAX),ARG(WA)
      I=1
      J=N
      K=0
      GT=1
1     LK=K
      K=(I+J)/2
      LF=IABS(K-LK)
      L=0
2     L=L+1
      IF(ARG(L) - TABLE(K,L)) 4,3,7
3     IF(L.LT.WA)     GO TO 2
      VALUE = TABLE(K,WT)
      RETURN 1
4     IF(TABLE(K,L).EQ.KBLA) GO TO 16
19    GO TO(13),LF
5     IF(K.LT.2) GO TO 13
6     J=K-1
      GO TO 1
7     IF(ARG(L).EQ.KBLA) GO TO 19
16    GO TO (9),LF
8     I=K+1
      GO TO 1
9     GO TO (10,12),GT
10    IF(K.GE.N) GO TO 12
      K=K+1
      GO TO 2
12    K=K+1
13    VALUE=K
      RETURN 2
      END
```

```
      SUBROUTINE BINS2(TABLE,IMAX,JMAX,N,ARG,VALUE,K,*,*)
      IMPLICIT INTEGER(A-W),LOGICAL(X-Z),REAL ($)
      DIMENSION TABLE(IMAX,JMAX)
      I=1
      J=N
      K=0
      GT=1
1     LK=K
      K=(I+J)/2
      LF=IABS(K-LK)
2     IF(ARG-TABLE(K,1)) 4,3,7
3     VALUE=TABLE(K,2)
      RETURN 1
4     GO TO (13),LF
5     IF(K-2) 13,6,6
6     J=K-1
      GO TO 1
7     GO TO (9),LF
8     I=K+1
      GO TO 1
9     GO TO (10,12),GT
10    IF(K-N) 11,12,12
11    GT=2
      K=K+1
      GO TO 2
12    K=K+1
13    VALUE=K
      RETURN 2
      END
```

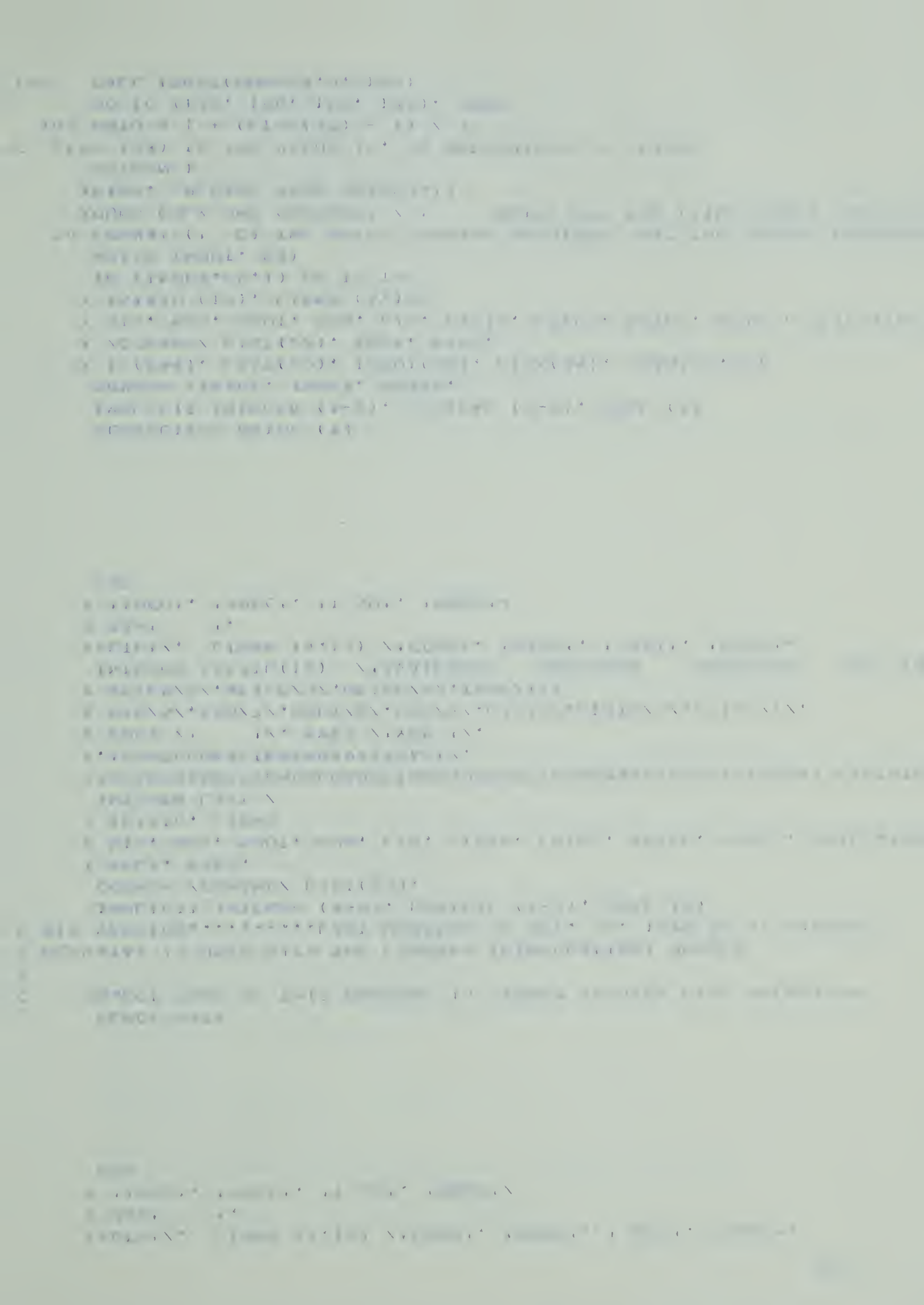
```
      SUBROUTINE BINS3 (ARG,WA,VALUE,V1,WV,K,*,*)
C MTS VERSION.........LAST UPDATED ON MAY 20, 1970 BY J. DIMSDALE
C
      IMPLICIT INTEGER(A-W),LOGICAL(X-Z),REAL($)
      COMMON XTRACE
      COMMON /PARAM/DUMMY(20),INDX25,MAXAUT
      DATA KBLA/'    '/ , DSRN/2 /, RSIZE/35/
      EQUIVALENCE (N,MAXAUT)
      DIMENSION TABLE(80),ARG(WA),VALUE(WV)
      IF(MAXAUT.EQ.0) WRITE(6,107)
107   FORMAT('  C) MAXAUT =0  IN BINS3.')
      I=1
      J=N
      K=0
      GT=1
1     LK=K
      K=(I+J)/2
      LF=IABS(K-LK)
43    FIND(DSRN'1000* K )
```



```
      INDX25 = K
      READ(DSRN,14) (TABLE(L),L=1,RSIZE)
14    FORMAT(80A1)
      IF(.NOT.XTRACE) WRITE(6,106) K, (TABLE(L),L=1,24)
106   FORMAT(1X,I5,3X,24A1)
2     DO 3 L = 1,WA
      IF(ARG(L)-TABLE(L))   4,3,7
3     CONTINUE
      DO 15 L = 1, WV
      VALUE(L) = TABLE(V1+L)
15    CONTINUE
      RETURN 1
4     IF(TABLE(L).EQ.KBLA) GO TO 16
19    IF(LF.EQ.1) GO TO 13
5     IF(K.LT.2) GO TO 13
6     J=K-1
      GO TO 1
7     IF(ARG(L).EQ.KBLA) GO TO 19
16    IF(LF.EQ.1) GO TO 9
8     I=K+1
      GO TO 1
9     GO TO (10,12),GT
10    IF(K.LT.N) GO TO 11
11    GT=2
      K=K+1
      GO TO 43
12    K=K+1
13    VALUE(1) = K
      RETURN 2
      END
```

```
      BLOCK DATA
C     OBJECT FORM OF THIS PROGRAM IS STORED IN DISK FILE  "BLKDATA3"
C
C BLKDATA3 IS USED WITH THE UPDATE MODULE
C MTS VERSION.........LAST UPDATED ON OCT. 20, 1970 BY J. DIMSDALE
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      COMMON /COMAND/ LIST(22),
     X KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU, LIBNM
      INTEGER LIST /
     X'SEARBRINRESEENQULOANRETUREQUDATADELEDEMASAVECOPYDISPDAY UPDASTOP'
     X,'ENROLOOKALTEREMOFINDTRAC'/,
     X KBLA /'    '/, KYES /'YES '/,
     X NIN/5/,KRD/7/,NOUT/6/,MON/3/,LIB/6/,LIBIN/5/,LFILE/1/,
     X MFILE/2/,NFILE/3/,OFILE/4/,TAPE/ 2/
      INTEGER ISTATU(12)  /'AVAILABLE   BORROWED    REQUESTED   SEE 1ST
```

```
     XAUTH'/,  LIBNM (4,10) /'COMP', 'UTER', ' SCI', 'ENCE',
     X 32*'    ',
     X 'INCO', 'RREC', 'T NU', 'MBER'/
      END



      BLOCK DATA
C     OBJECT FORM OF THIS PROGRAM IS STORED IN DISK FILE "BLKDAT4"
C
C BLKDATA4 IS USED WITH THE LIBRARY (CIRCULATION) MODULE
C MTS VERSION.........LAST UPDATED ON OCT. 20, 1970 BY J. DIMSDALE
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      COMMON /COMAND/ LIST(22),
     X KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU, LIBNM
      INTEGER LIST /
     X'SEARBRINRESEENQULOANRETUREQURENEFINDDEMASAVECOPYDISPDAY LISTSTOP'
     X,'ENROLOOKALTEREMOAVAITRAC'/,
     X KBLA /'    '/, KYES /'YES '/,
     X NIN/5/,KRD/7/,NOUT/6/,MON/8/,LIB/6/,LIBIN/5/,LFILE/1/,
     X MFILE/2/,NFILE/3/,OFILE/4/,TAPE/11/
      INTEGER ISTATU(12)  /'AVAILABLE   BORROWED    REQUESTED   SEE 1ST
     XAUTH'/,  LIBNM (4,10) /'COMP', 'UTER', ' SCI', 'ENCE',
     X 32*'    ',
     X 'INCO', 'RREC', 'T NU', 'MBER'/
      END




      SUBROUTINE BRING (*)
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      IF (INDEX.EQ.1) GO TO 100
      WRITE (NOUT, 50)
   50 FORMAT ('  C) THE BRING COMMAND REQUIRES THAT THE SEARCH COMMAND F
     XOUND ONLY ONE CORRECT' / '     ENTRY FOR THE TITLE WORDS YOU SPECI
     XFIED.  PLEASE TYPE SEARCH.')
      RETURN 1
C  LINK (33) IS THE VALUE IC, AS DETERMINED BY STATUS
  100 GOTO = 1 + (LINK(33) - 1) / 3
      GO TO (105, 150, 160, 170), GOTO
105   CALL IDENT(NUMBER,0,&180)
```

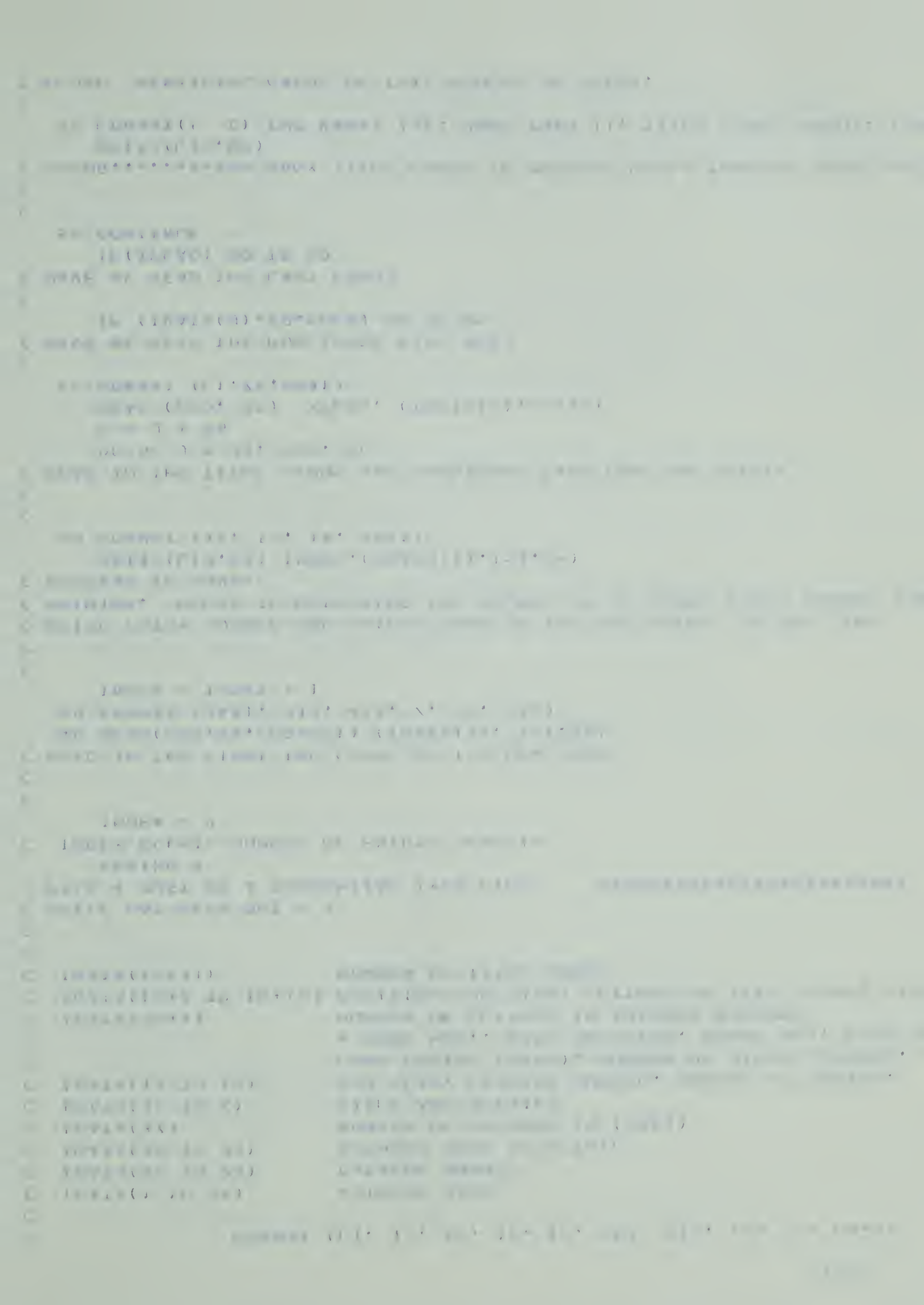


```
      WRITE (NOUT, 117) (LINK(I), I = 29, 32)
  117 FORMAT ('  C) COLLECT FROM ',4A4,' LIBRARY WITHIN THIRTY MINUTES')
      WRITE (LIB, 121) (LINK(I), I = 1, 28), NUMBER
  121 FORMAT (5X, 'C) LIBRARY BRING ', 24A1, 4A4, ' FOR ID ', I6)
      RETURN 1
  150 WRITE (NOUT, 155)
  155 FORMAT ('  C) THIS BOOK IS NOT AVAILABLE.  TYPE RESERVE')
      RETURN 1
  160 WRITE (NOUT, 165)
  165 FORMAT ('  C) THIS BOOK IS NOT AVAILABLE, NOR CAN IT ACCEPT ANY MO
     XRE REQUESTS.  SORRY.')
      RETURN 1
  170 WRITE (NOUT, 175)
  175 FORMAT ('  C) TYPE SEARCH AND GIVE THE NAME OF THE FIRST AUTHOR LI
     XSTED AFTER THE TITLE AS YOUR RESPONSE.')
180   RETURN 1
      END




      SUBROUTINE CHANGE(PASS1,PASS2)
C
C
C THIS SUBROUTINE  MAY BE USED TO CHANGE OR DELETE ANY ONE OR MORE
C OF THE FOUR CHARACTERS STORED IN AN INTEGER*4 WORD.
C PASS1 = WORD TO BE CHANGED
C PASS2 = WORD CONTAINING THE REQUIRED CHANGES:
C                   1) TO CHANGE A CHARACTER IN PASS1 PUT ITS REPLACEMENT
C                      IN THE CORRESPONDING POSITION OF PASS2
C                   2) TO DELETE A CHARACTER IN PASS1 PUT A "*" IN THE
C                      CORRESPONDING POSITION IN PASS2
C                   3) TO LEAVE A CHARACTER IN PASS1 UNCHANGED LEAVE THE
C                      CORRESPONDING POSITION IN PASS2 BLANK
C
      IMPLICIT INTEGER(A-W), LOGICAL(X-Z),REAL($)
      DIMENSION TEMP(4)
      LOGICAL*1 XOLD(4),XNEW(4),XTEMP(16)
      EQUIVALENCE(OLD, XOLD(1)),(NEW,XNEW(1)),(TEMP(1),XTEMP(1))
      DATA KBLA/'    '/,STAR/'*   '/
C
      NEW = PASS2
      OLD = PASS1
      DO 2 I =1,4
2     TEMP(I) = KBLA
      DO 5 I=1,4
5     XTEMP(4*I-3) = XNEW(I)
      DO 10 I=1,4
      IF(TEMP(I).EQ.KBLA)   GO TO 10
      IF(TEMP(I).EQ.STAR) TEMP(I) = KBLA
      XOLD(I) = XTEMP(4*I-3)
10    CONTINUE
```

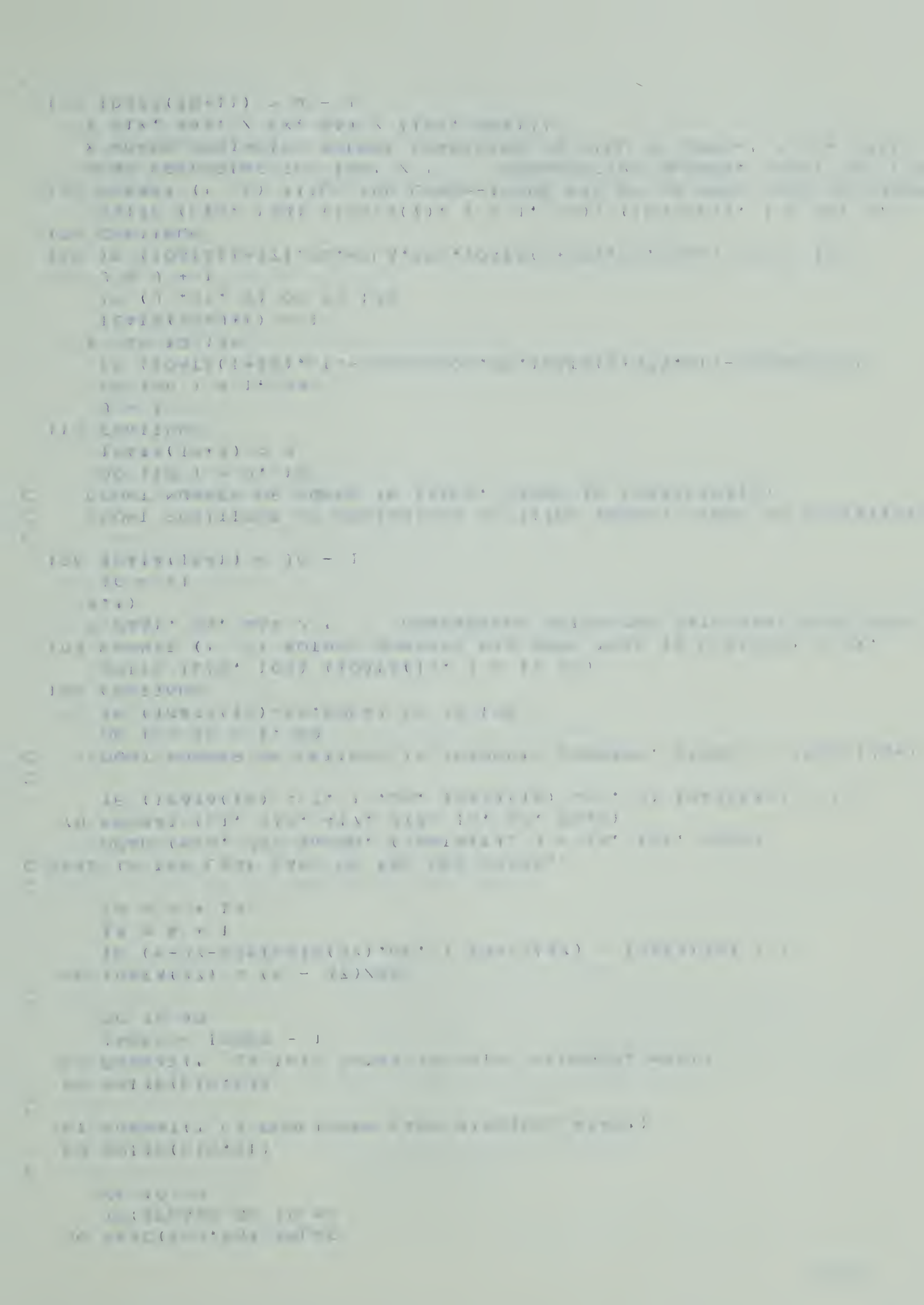
```
      PASS1 = OLD
      RETURN
      END



      SUBROUTINE CHECK (DEVICE, LINE, START, STOP, RESULT, MODE, *)
C
C VERSION 1.5........JJFD.......JULY 9/69
C REWRITTEN SECTION IS BETWEEN THE    ********* LINES
C
C  THIS SUBROUTINE CHECKS FOR LEFT JUSTIFICATION AND POSITION OF
C  RIGHT MOST CHARACTER OF LINE.  IT WILL CONVERT EBCDIC TO BINARY.
C
C  MODE IS .TRUE. IF ONLY LEFT JUSTIFICATION IS TO BE CHECKED.
C
C     ONMZE = '1    ' - '0    '
      INTEGER BLANK/'     '/,ZERO/'0    '/,NINE/'9    '/,ONMZE/16777216/,
     X RESULT, START, STOP, DEVICE, LINE(1)
      INTEGER RESTOR /Z14000000/
      LOGICAL MODE , SWITCH
      IF (LINE(START) / 16777216 - 64) 1, 4, 1
    1 IF (MODE) RETURN
C
C ***********
      SWITCH = .TRUE.
      GO TO 2
C
C
      ENTRY CHECKR(DEVICE,LINE,START,STOP,RESULT,*)
C
C
C CHECKR CHECKS ONLY FOR THE POSITION OF THE RIGHT MOST NONBLANK
C CHARACTER OF LINE BEFORE CONVERTING EBCDIC TO BINARY.  IF LINE IS
C COMPOSED ENTIRELY OF BLANKS THEN CHECKR RETURNS RESULT = 0.
C
      SWITCH = .FALSE.
C
    2 DO 3 J = START, STOP
      IF (LINE(STOP+START-J) .NE. BLANK) GO TO 8
    3 CONTINUE
      IF (SWITCH)  GO TO 4
      RESULT = 0
      RETURN
C ***********
C
    4 WRITE (DEVICE,5) RESTOR
    5 FORMAT (2X, A1, 'C) RESPONSE MUST BE LEFT JUSTIFIED')
      RETURN 1
    8 LAST = STOP + START - J
C
```

```
C
      ENTRY EBCDIC (DEVICE, LINE, START, LAST, RESULT, *)
C
C
      RESULT = 0
      DO 10 I = START, LAST
      IF (LINE(I).EQ.BLANK) LINE(I) = ZERO
      IF (LINE(I).GT.NINE .OR. LINE(I).LT.ZERO) GO TO 20
      RESULT = 10*RESULT + (LINE(I) - ZERO)/ONMZE
   10 CONTINUE
      RETURN
   20 WRITE (DEVICE,30) RESTOR, LINE(I)
   30 FORMAT (2X, A1, 'C) INVALID CHARACTER..', A1, '      ')
      RETURN 1
      END



      SUBROUTINE DATA(*)
C
C MTS VERSION ..........LAST UPDATED ON AUGUST 2, 1970 BY J. DIMSDALE
C THIS VERSION INTENDED TO BE USED IN BATCH MODE ONLY.
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL($)
      COMMON XTRACE, INDEX, NDATE,
     X IDATA(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
C
C        THIS SUBROUTINE READS DATA FROM CARDS AND AFTER SOME PROCESSING
C        AND REARRANGEMENT WRITES IT ON NFILE.
C THE CARDS ARE FOUND IN FILE 'NEWBOOKS' AND NFILE = 'NEWENTRIES'
C KRD = 'NEWBOOKS'
C
C    CARD FORMAT:
C    CARD 1      AUTHOR SURNAME AND INITIALS (24A1), 21X, CATALOG NUMBER
C    CARD 2      AUTHOR SURNAME AND FULL GIVEN NAMES (INITIALS) (8A4)
C    CARDS 3 - , TITLE AND OTHER DETAILS (8X, 66A1).  THE PART OF THE TI
C                BE SEARCHED (AS MANY AS 9 WORDS) COMES FIRST AND MUST BE
C                SEPARATED FROM THE REMAINDER BY 2 OR MORE BLANKS.  INTEG
C                AND PUNCTUATION MARKS WILL BE IGNORED.  AS MANY AS 11 SU
C                MAY BE USED.
C    CARD        PUBLISHER, PLACE OF PUBLICATION, DATE, NUMBER OF PAGES
C                ONE CARD, (10X, 64A1)
C    LAST CARD   END FLAG ("F" IF LAST TITLE, "T" OTHERWISE);
C                NUMBER OF LIBRARY CONTAINING BOOK; NUMBER OF COPIES
C                (IF 0, MEANS THAT TRANSACTIONS ARE HANDLED UNDER THE
C                FIRST AUTHOR LISTED AFTER THE TITLE); LOAN PERIOD IN DAY
C                NO. OF OTHER AUTHORS, AS MANY AS 4 BOOK NUMBERS,
C                ACQUISITION DATE, ORDER NUMBER,, FUND NO.,, COST.
```

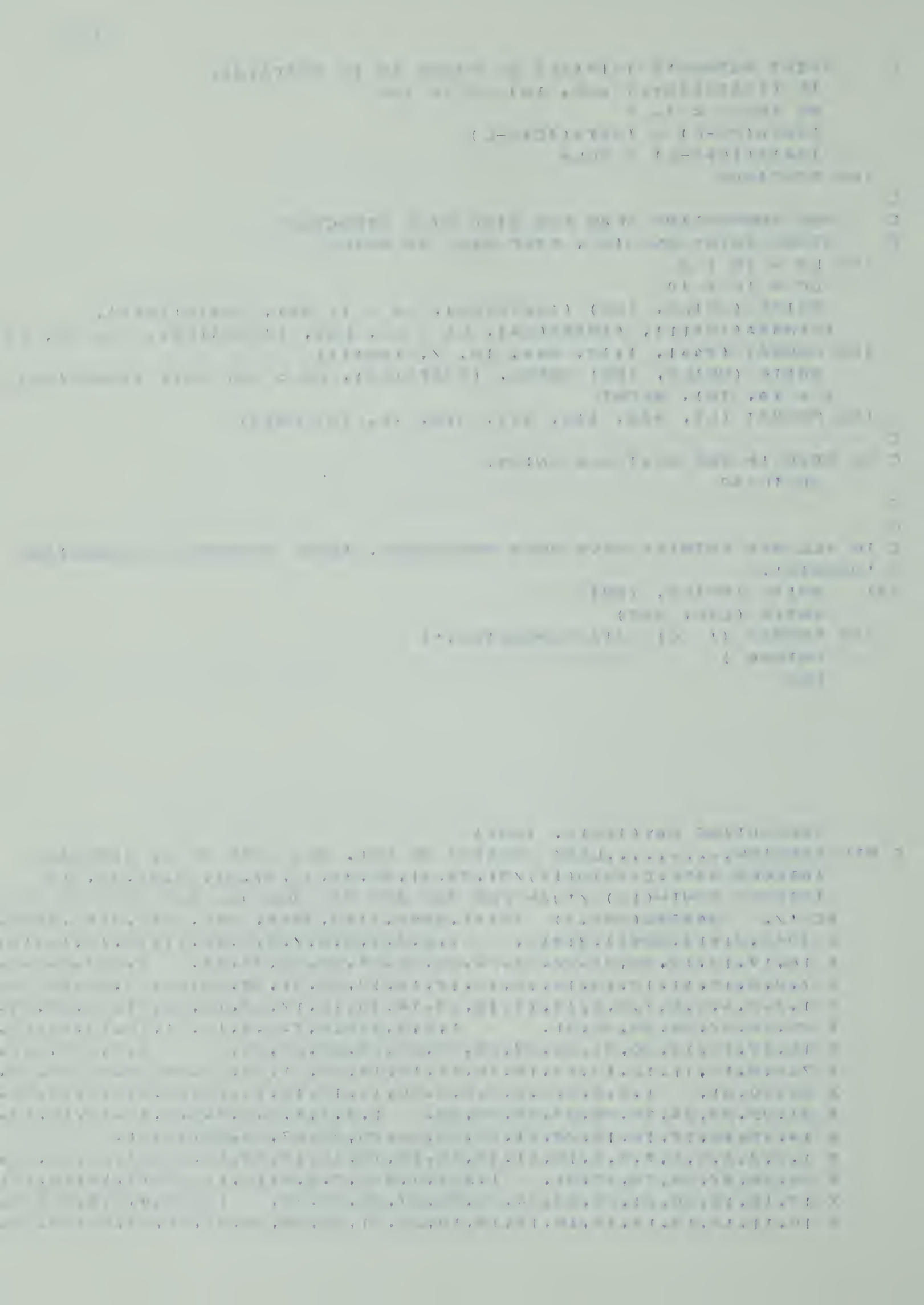
```
C                FORMAT (L1, I2, I2, I2, I2, 4I7, 3I2, I6, I2, F6.2)
C
C  IDATA(1 TO 24)         AUTHORS NAME
C  IDATA(25 TO 28)        CATALOG NUMBER
C  IDATA(29 TO 36)        AUTHORS NAME IN DETAIL
C  IDATA(37)              NUMBER OF RECORDS IN (80A1)
C  IDATA(38 TO K)         TITLE AND DETAILS
C  IDATA(IA TO IB)        END FLAG, LIBRARY NUMBER, NUMBER OF COPIES,
C                         LOAN PERIOD (DAYS), NUMBER OF OTHER AUTHORS,
C                         4 BOOK NOS., DATE RECEIVED, ORDER NO., FUND NO.,
C  IDATA(IB+1)            NUMBER OF LETTERS IN AUTHORS SURNAME
C  IDATA(IB+2 TO IB+10) POSITIONS OF FIRST LETTERS OF TITLE WORDS WITH 1
C  IDATA(IB+11)           NUMBER OF TITLE WORDS
C
C
C NFILE HAS BEEN SET = 4
C FILE 4 MUST BE A SEQUENTIAL TYPE FILE      **************************
      REWIND 4
C  INDEX COUNTS NUMBER OF ENTRYS READ IN
      INDEX = 0
C
C
C READ IN THE FIRST TWO CARDS OF THE NEW ENTRY
   40 READ(KRD,50,END=221) (IDATA(I), I=1,36)
   50 FORMAT (24A1, 21X, 4A4, /, 6X, 8A4)
      INDEX = INDEX + 1
C
C
C WRITE ENTRY NUMBER AND AUTHOR NAME OF THE NEW ENTRY  ON THE LINE
C PRINTER.  HELPS TO DETERMINE THE SOURCE OF AN ERROR WHICH CAUSES THE
C PROGRAM TO BOMB.
      WRITE(LIB,53) INDEX,(IDATA(I),I=1,24)
   53 FORMAT (1X, I3, 1X, 24A1)
C
C
C READ IN THE TITLE CARDS AND PUBLISHER CARD FOR NEW ENTRY.
      DO 56 J = 38, 698, 66
      K = J + 65
      READ (KRD, 55)  XFLAG, (IDATA(L),L=J,K)
   55 FORMAT (L1,7X,66A1)
C
C HAVE WE READ THE PUBLISHER CARD YET?
      IF (IDATA(J).EQ.KBLA) GO TO 65
C
C HAVE WE READ THE LAST CARD?
      IF(XFLAG) GO TO 60
   56 CONTINUE
C
C
C ERROR.......TOO MANY TITLE CARDS IF PROGRAM FALLS THROUGH ABOVE LOOP.
      WRITE(LIB,58)
   58 FORMAT('  C) TOO MANY, I.E. MORE THAN 11, TITLE CARDS INPUT. ***')
C
C FLUSH  REMAINING CARDS OF THIS FOULED UP ENTRY.
```



```
   59 READ(KRD,55) XFLAG
      IF(XFLAG) GO TO 62
      GO TO 59
C
   60 WRITE(LIB,61)
   61 FORMAT(' C) PUBLISHER CARD MISSING. *****')
C
   62 WRITE(LIB,63)
   63 FORMAT('  C) THIS ENTRY IS BEING FLUSHED. ***')
      INDEX = INDEX - 1
      GO TO 40
C
   65 IDATA(37) = (K - 37)/80
      IF (K-37-80*IDATA(37).NE.0) IDATA(37) = IDATA(37) + 1
      IA = K + 1
      IB = K + 13
C
C READ IN THE LAST CARD OF THE NEW ENTRY.
      READ (KRD, 70) XNEND, (IDATA(I), I = IA, IB), $COST
   70 FORMAT (L1, 4I2, 4I7, 3I2, I6, I2, F6.2)
      IF (IDATA(IA) .LT. 1 .OR. IDATA(IA) .GT. 9) IDATA(IA) = 10
C
C    COUNT NUMBER OF LETTERS IN AUTHOR'S SURNAME, STORE IN IDATA(IB+1)
      DO 100 IC = 1, 20
      IF (IDATA(IC).EQ.KBLA) GO TO 106
  100 CONTINUE
      WRITE (LIB, 103) (IDATA(I), I = 1, 28)
  103 FORMAT ('  C) AUTHOR SURNAME HAS MORE THAN 19 LETTERS' / 5X,
     X 24A1, 5X, 4A4 / '      CHARACTERS AFTER THE 24TH HAVE BEEN DELETED
     X.')
      IC = 21
  106 IDATA(IB+1) = IC - 1
C
C    COUNT POSITIONS OF BEGINNINGS OF TITLE WORDS. STORE IN IDATA(IB+2 T
C    COUNT NUMBER OF WORDS IN TITLE, STORE IN IDATA(IB+11)
      DO 115 I = 2, 10
      IDATA(IB+I) = 0
  115 CONTINUE
      J = 1
      DO 130 I = 1, 99
      IF (IDATA(I+36).LT.-300000000.OR.IDATA(I+37).GT.-300000000)
     X  GO TO 125
      IDATA(IB+J+1) = I
      IF (J .GT. 9) GO TO 135
      J = J + 1
  125 IF (IDATA(I+37).EQ.KBLA.AND.IDATA(I+38).EQ.KBLA) GO TO 135
  130 CONTINUE
      WRITE (LIB, 133) (IDATA(I), I = 1, 28), (IDATA(I), I = 38, K)
  133 FORMAT ('  C) TITLE TOO LONG--THERE CAN BE NO MORE THAN 98 CHARACT
     XERS PRECEDING THE TWO' / '      CONSECUTIVE BLANKS.  ONLY THE TITLE
     X WORDS BEGINNING BEFORE CHARACTER 99 WILL BE USED.' / 5X, 24A1,
     X 21X, 4A4, / 6X, 8A4 / (13X, 66A1))
  135 IDATA(IB+11) = J - 1
C
```

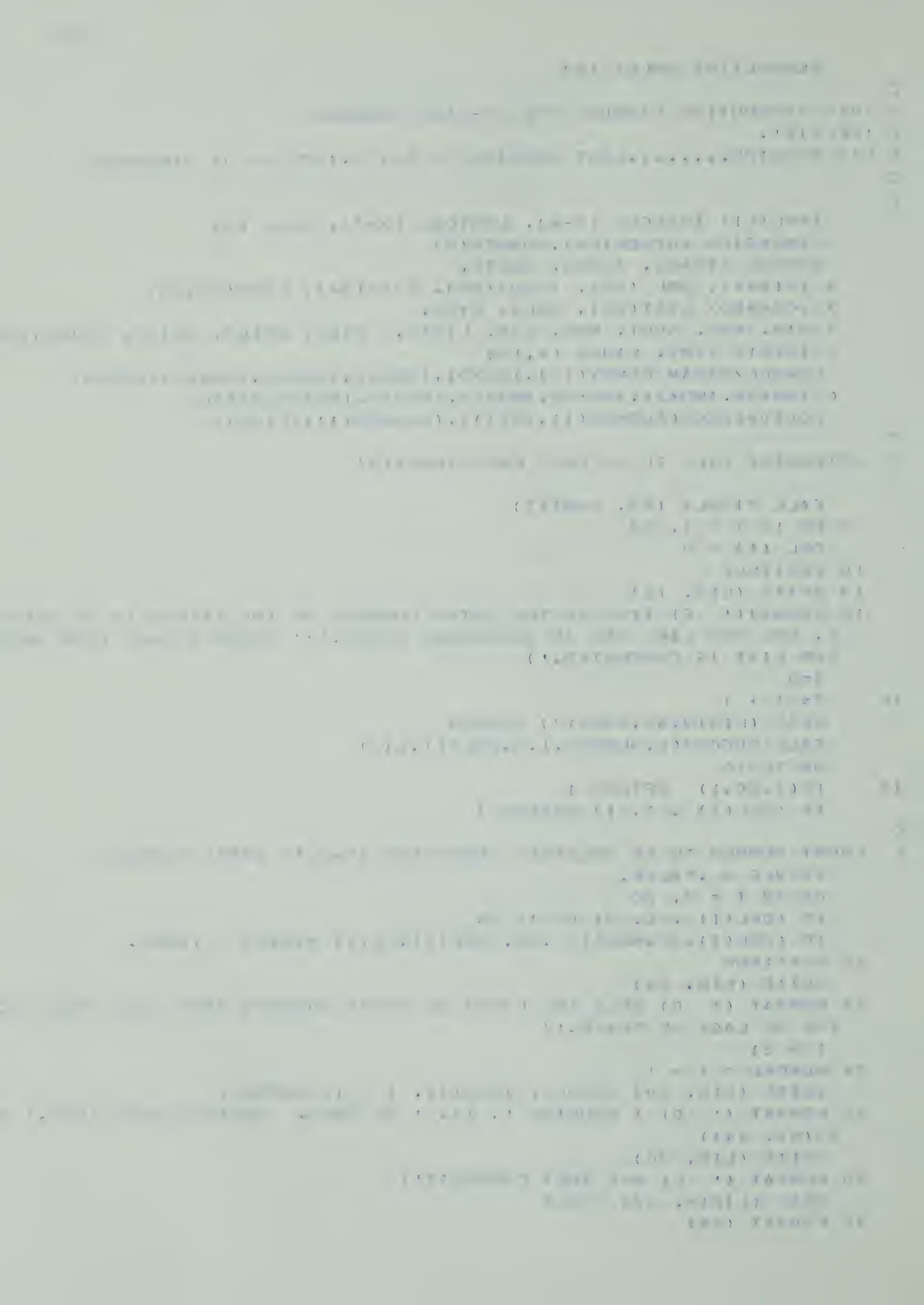


```
C     SHIFT AUTHOR'S INITIALS SO FIRST IS IN IDATA(21)
      IF (IDATA(IB+1) .GE. 19) GO TO 165
      DO 155 L = 1, 4
      IDATA(25-L) = IDATA(IC+5-L)
      IDATA(IC+5-L) = KBLA
  155 CONTINUE
C
C     SEE SUBROUTINE SEAR FOR DISC FILE STRUCTURE
C     STORE ENTRY ON DISC, FILE NAME IS NFILE
  165 LB = IB + 2
      LC = IB + 10
      WRITE (NFILE, 180) (IDATA(LA), LA = 1, 24), IDATA(IB+1),
     X IDATA(IB+11), (IDATA(LA), LA = LB, LC), (IDATA(LA), LA= 29, K)
  180 FORMAT (24A1, 11I2, 8A4, I2, /, (80A1))
      WRITE (NFILE, 182) XNEND, (IDATA(LA), LA = 25, 28), (IDATA(LA), LA
     X = IA, IB), $COST
  182 FORMAT (L1, 4A4, 4I2, 4I7, 3I2, I6, I2, F6.2)
C
C GO READ IN THE NEXT NEW ENTRY.
      GO TO 40
C
C
C IF ALL NEW ENTRIES HAVE BEEN PROCESSED, THEN  RETURN TO SUBROUTINE
C 'UPDATE'.
221   WRITE (MFILE, 182)
      WRITE (LIB, 222)
  222 FORMAT ('  C) DATA COMPLETED.')
      RETURN 1
      END



      SUBROUTINE DAY(IDATE, IOUT)
C MTS VERSION..........LAST UPDATED ON OCT. 20, 1970 BY J. DIMSDALE
      INTEGER DATE,DAYSIN(12)/31,28,31,30,31,30,31,31,30,31,30,31/
      INTEGER MONTH(12) /'JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DE
     XC '/,   DAYS*2(365,2)  /31*1,28*2,31*3,30*4,31*5,30*6,31*7,31*8,
     X 30*9,31*10,30*11,31*12,    1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
     X 16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,   1,2,3,4,5,6,
     X 7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,
     X 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
     X 25,26,27,28,29,30,31,      1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
     X 16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,      1,2,3,4,5,6,
     X 7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,
     X 29,30,31,   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
     X 21,22,23,24,25,26,27,28,29,30,   1,2,3,4,5,6,7,8,9,10,11,12,13,
     X 14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
     X 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
     X 25,26,27,28,29,30,31,   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,
     X 17,18,19,20,21,22,23,24,25,26,27,28,29,30,   1,2,3,4,5,6,7,8,9,
     X 10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,
```

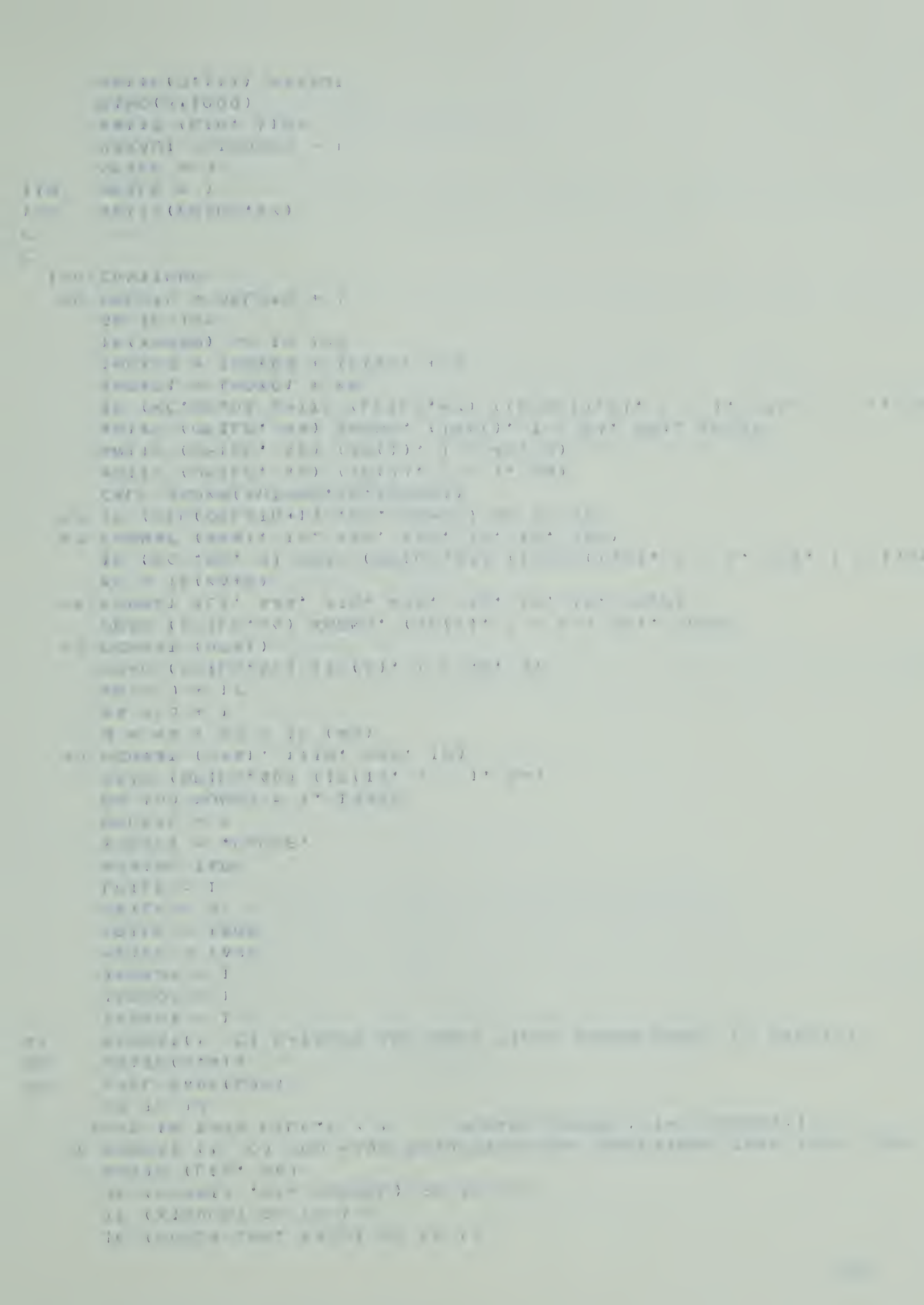


```
     X 31,   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
     X 23,24,25,26,27,28,29,30,   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
     X 16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31/, IOUT(3)
C
C  THIS PROGRAM DETERMINES THE MONTH NAME, THE DAY OF THE MONTH AND THE
C  YEAR FROM AN INTEGER CALLED IDATE.  WHEN IDATE = 1, THE DATE IS
C  JAN 1, 1968.  THE 3 COMPONENTS OF THE DATE ARE PLACED IN IOUT.
C
C  FIND YEAR
      INPUT = IDATE
      J = 0
      DO 10 I = 1, 8
      IF (INPUT.LE.366) GO TO 12
      INPUT = INPUT - 366
      DO 8 J = 1, 3
      IF (INPUT.LE.365) GO TO 12
      INPUT = INPUT - 365
    8 CONTINUE
   10 CONTINUE
   12 IOUT(3) = 64 + 4 * I + J
C
C  CHECK FOR LEAP YEAR
      IF (IOUT(3)-4*(IOUT(3)/4) .NE. 0) GO TO 14
C
C  CHECK FOR FEB 29 AND DATE AFTER FEB 29
      IF (INPUT - 60) 14, 16, 13
   13 INPUT = INPUT - 1
   14 IOUT(1) = MONTH(DAYS(INPUT,1))
      IOUT(2) = DAYS(INPUT,2)
      RETURN
   16 IOUT(1) = MONTH(2)
      IOUT(2) = 29
      RETURN
C
      ENTRY DATE(IDATE,IOUT)
C
C GIVEN THE DATE IN FORM 10:29:70, CALCULATE
C DAY NUMBER RELATIVE TO JAN. 1, 1968 AS DAY 1.
C IOUT(1) CONTAINS MONTH OF CURRENT DATE.
C IOUT(2) CONTAINS DAY OF CURRENT DATE.
C IOUT(3) CONTAINS YEAR OF CURRENT DATE.
C
      IDATE=IOUT(2)+365*(IOUT(3)-68)
      J=IOUT(1)-1
      DO 20 I=1,J
20    IDATE=IDATE+DAYSIN(I)
C ACCOUNT FOR LEAP YEARS.
      J=IOUT(3)
      DO 23 I=68,J,4
23    IDATE=IDATE+1
      IOUT(1)=MONTH(IOUT(1))
      RETURN
      END
```

```
      SUBROUTINE DELET (*)
C
C THIS SUBROUTINE HANDLES THE ON-LINE COMMAND
C 'DELETE'.
C MTS VERSION........LAST UPDATED ON MAY 9,1970 BY J. DIMSDALE
C
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      DIMENSION AUTHOR(24),NUMBER(5)
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), DEL (20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      COMMON/PARAM/DUMBY(10),INDX01,INDX02,INDX03,INDX04,INDX08,
     C INDX20,INDX21,INDX22,INDX23,INDX24,INDX25,MAXAUT
      EQUIVALENCE(AUTHOR(1),IF(1)),(NUMBER(1),IF(25))
C
C  DETERMINE FILE TO BE USED FROM INPUT(3)
C
      CALL DISPLA (&9, NUMTIT)
    9 DO 10 I = 1, 20
      DEL (I) = 0
   10 CONTINUE
   13 WRITE (LIB, 15)
   15 FORMAT('  C) TYPE IN THE ENTRY NUMBERS OF THE TITLES TO BE DELETED
     C, ONE PER LINE AND IN ASCENDING ORDER.'/' ENTER A NULL LINE WHEN T
     CHE LIST IS COMPLETED.')
      I=0
16    I= I + 1
      READ (LIBIN,40,END=17) NUMBER
      CALL CHECKR(6,NUMBER,1,5,DEL(I),&13)
      GO TO 16
17    IF(I.EQ.1)  RETURN 1
      IF (DEL(1) .LT. 1) RETURN 1
C
C  COUNT NUMBER TO BE DELETED; CHECK FOR INVALID ENTRY NUMBERS
      XINVLD = .FALSE.
      DO 22 I = 2, 20
      IF (DEL(I) .EQ. 0) GO TO 24
      IF (DEL(I).GE.NUMTIT .OR. DEL(I).LT.1) XINVLD = .TRUE.
   22 CONTINUE
      WRITE (LIB, 23)
   23 FORMAT ('  C) ONLY THE FIRST 20 ENTRY NUMBERS HAVE BEEN USED BECAU
     XSE OF LACK OF SPACE.')
      I = 21
   24 NUMDEL = I - 1
      WRITE (LIB, 28) NUMDEL, (DEL(I), I = 1, NUMDEL)
   28 FORMAT ('  C) I COUNTED ', I2, ' OF THEM.  PLEASE CHECK THEM.' /
     X (5X, I4))
      WRITE (LIB, 30)
   30 FORMAT ('  C) ARE THEY CORRECT?')
      READ (LIBIN, 32) REPLY
   32 FORMAT (A4)
```



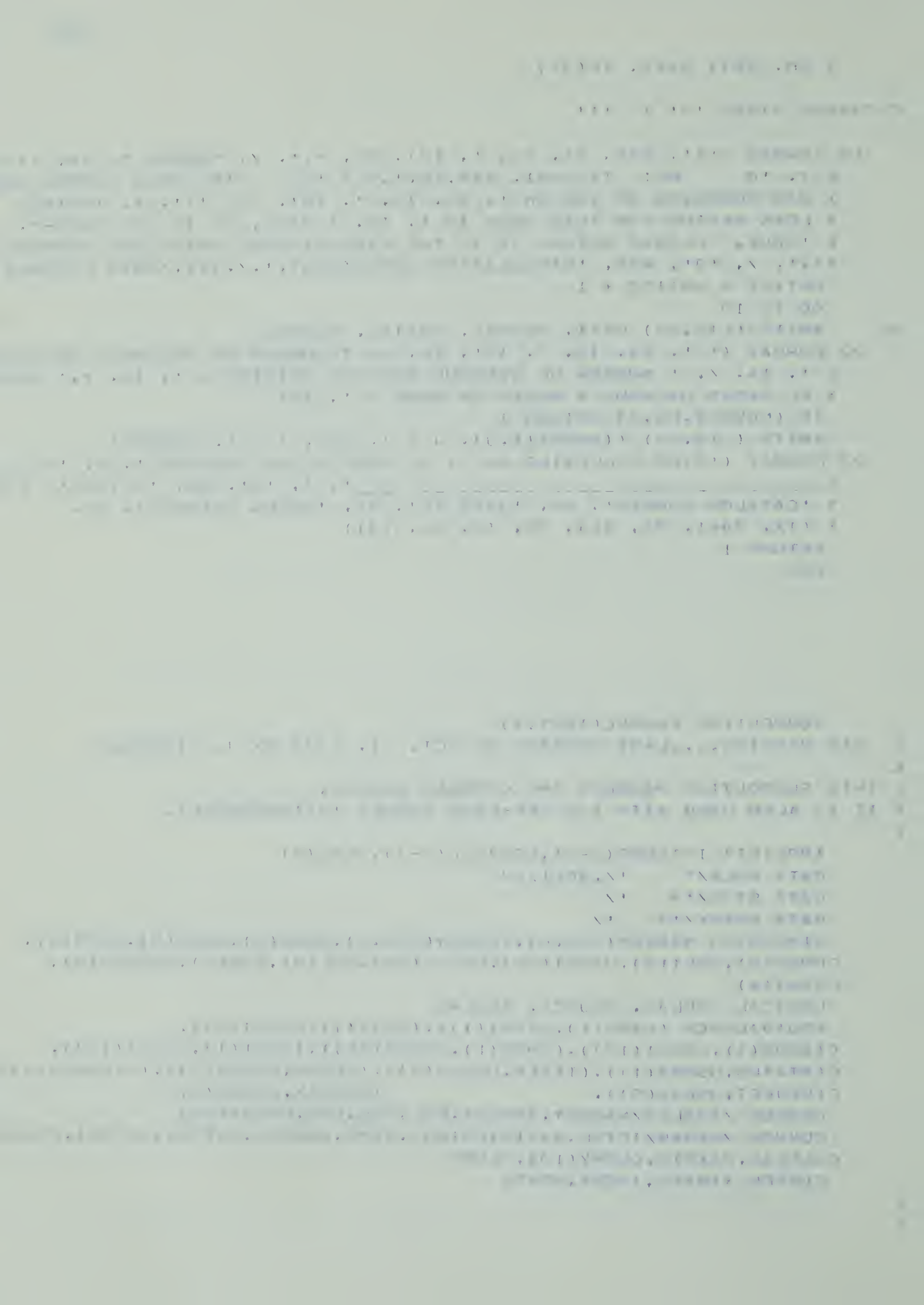
```
      IF (REPLY .NE. KYES) GO TO 13
      IF (XINVLD) GO TO 130
      IF (NUMTIT .GT. NUMDEL) GO TO 38
      WRITE (LIB, 36)
   36 FORMAT ('  C) YOU HAVE REQUESTED MORE DELETIONS THAN THERE ARE ENT
     XRYS IN THIS FILE.' / '      PLEASE CORRECT THE ERROR.')
      GO TO 13
38    CALL SAVE(&39)
39    WRITE(6,41)
41    FORMAT('  C) CATALOG AND LOAN FILES TRANSFERRED TO TAPE.')
      INDX01 = 1
      INDX03 = 1
      INDX25 = 1
      MFILE = TAPE
      NFILE = TAPE
      OFILE = 3
      LFILE = 1
      REWIND TAPE
      XRESET = .FALSE.
      DELETD = 0
      DO 100 NUMRD = 1, 10000
      READ (NFILE,40) (IF(I), I = 1, 44)
   40 FORMAT (24A1, 11I2, 8A4, I2)
      J = 44 + 80 * IF (44)
      KA = J + 1
      KB = J + 17
      READ (NFILE,42) (IF(I), I = 45, J)
   42 FORMAT (80A1)
      READ (NFILE,44) XNEND, (IF(I), I = KA, KB), $COST
   44 FORMAT (L1, 4A4, 4I2, 4I6, 3I2, I6, I2, F6.2)
      KC = IF(KA+5)
      IF (KC .NE. 0) READ (MFILE,47) ((LOAN(I,K), K = 1, 37), I = 1,KC)
   47 FORMAT (24A1, I2, 4A4, 4I6, I4, I3, 2I2)
   53 IF (DEL(DELETD+1) .EQ. NUMRD) GO TO 95
      CALL INDXR(AUTHOR,KC,XRESET)
      WRITE (OFILE, 40) (IF(I), I = 1, 44)
      WRITE (OFILE, 42) (IF(I), I = 45, J)
      WRITE (OFILE, 44) XNEND, (IF(I), I = KA, KB), $COST
      IF (KC.NE.0) WRITE (LFILE,47) ((LOAN(I,K), K = 1, 37), I = 1, KC)
      INDX01 = INDX01 + KC
      INDX03 = INDX03 + IF(44) + 2
      IF(XNEND) GO TO 100
      GO TO 105
   95 DELETD = DELETD + 1
  100 CONTINUE
C
C
105   WRITE(LFILE,47)
110   MFILE = 1
      NFILE = 3
      MAXAUT = INDX25 - 1
      WRITE (LIB, 112)
      FIND(3'1000)
      WRITE(3,111) MAXAUT
```



```
111   FORMAT(I5)
 112  FORMAT ('  C) DELETE COMPLETED.')
      RETURN 1
 130  WRITE (LIB, 132)
 132  FORMAT ('  C) DELETION OF ENTRYS WITH NEGATIVE ENTRY NUMBERS OR WI
     XTH ENTRY NUMBERS' / '      EXCEEDING THAT OF THE LAST DELETEABLE EN
     XTRY IS NOT POSSIBLE.')
      GO TO 13
      END



      SUBROUTINE DEMAND (*)
C     MTS VERSION.....    LAST UPDATED ON SEPT. 30, 1970 BY J. DIMSDALE
C    THIS SUBROUTINE  CHECKS FOR OVERDUE BOOKS AND PRINTS OVERDUE NOTICES
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      DIMENSION MORE4(50,30), DAY1(3), DAY2(3)
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      COMMON/USER/NAME(5),STATUS,TITLE,ADDR(6),PHON(2),DEPT(2),IDNUM,K1,
     CK2,ACCT,EDATE,ONLOAN,OVRDUE,OVER1
      COMMON/PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL,USTOR1,OSTOR1,FSIN24,
     CALFLAG,MAXPTR,INDX01,INDX02,
     CINDX03,INDX04,INDX08,INDX20,INDX21,INDX22,INDX23,INDX24
      DIMENSION CATLOG(4)
      EQUIVALENCE (CATLOG(1),IF(26)),(LIBID,IF(30)),(LDATE,IF(34)),
     C(WEEKS,NWEEK)
C
C  STRUCTURE OF MFILE
C     (24A1)    AUTHOR
C     (I2)      LENGTH OF AUTHOR SURNAME
C     (4A4)     CATALOG NUMBER
C     (4I6)     ID NUMBERS OF 4 USERS
C     (I4)      DAY FIRST USER BORROWED BOOK.  (IF BOOK NOT OUT, DAY
C               NUMBER IS DAY BOOK BECAME AVAILABLE TO FIRST REQUESTOR
C     (I3)      NUMBER OF TIMES THIS COPY HAS BEEN BORROWED INCLUDING PRES
C     (I2)      NUMBER OF COPIES
C     (I2)      NUMBER OF DAYS BOOK MAY BE BORROWED
C
C RESET RESERVE-DELETIONS COUNTER
      NUMDEL = 0
C RESET OVERDUE-NOTICE COUNTER
      NOTICE = 0
C RESET MORE-THAN-4-WEEKS OVERDUE COUNTER
      NOVER4 = 0
      CALL DAY (NDATE, DAY1)
      REWIND 7
```

```
      MFILE = 1
C SET LOAN FILE POINTER TO START OF LOANS FILE
5     INDX01 = 1
      FIND(MFILE*1000*INDX01)
C READ A LOAN RECORD
   10 READ (MFILE, 11,END= 20) (IF(I), I= 1, 37)
   11 FORMAT (24A1, I2, 4A4, 4I6, I4, I3, 2I2)
C INCREMENT LOANS FILE POINTER
      INDX01 = INDX01 + 1
C END OF LOANS FILE?
      IF (IF(1).EQ.KBLA) GO TO 20
C IS THE BOOK  OUT?    IF IT IS THEN FIND OUT IF IT IS OVERDUE.
      IF (IF(30).NE.0) GO TO 14
C IS THERE A RESERVE QUEUE?        SHOULD IT BE MOVED UP  BY DELETING
C THE FIRST  RESERVATION
      IF (IF(31).EQ.0.OR.NDATE-IF(34).LE.3) GO TO 10
C IF BOOK HAS BEEN AVAILABLE TO THE FIRST REQUESTER FOR MORE THAN 3
C DAYS THEN  SHIFT UP THE QUEUE
      IF(31) = IF(32)
      IF(32) = IF(33)
      IF(33) = 0
      IF(34) = NDATE
      NUMDEL = NUMDEL + 1
C WRITE CHANGED LOAN RECORD BACK INTO THE LOANS FILE
      FIND(MFILE*1000*(INDX01-1))
   12 WRITE (MFILE,11) (IF(I), I = 1, 37)
      GO TO 10
C
C       NWEEK:  NUMBER OF WEEKS OVERDUE, TRUNCATED TO INTEGER
   14 NWEEK = (NDATE - IF(34) - IF(37))/7
C
C
C UPDATE USER'S RECORD
C     IF(NWEEK.GT.0) CALL OBOOK(LIBID,CATLOG,LDATE,WEEKS)  *** REMOVED *
C
C
      IF (NWEEK.LT.4) GO TO 17
C
C       STORE PART OF ENTRY TO BE OUTPUT WHEN FINISHED CHECKING FILE
      NOVER4 = NOVER4 + 1
      DO 15 I = 1, 24
      MORE4(NOVER4,I) = IF(I)
   15 CONTINUE
      DO 16 I = 26, 30
      MORE4(NOVER4,I-1) = IF(I)
   16 CONTINUE
      MORE4(NOVER4,30) = NWEEK
C
C       TO REQUIRE NOTICE, BOOK MUXT BE DUE AN EXACT WHOLE NUMBER OF WEEK
   17 IF (NDATE-IF(34)-IF(37)-7*NWEEK.NE.0.OR.NWEEK.LT.2) GO TO 10
C
C       OUTPUT NOTICE TO USER
      CALL DAY (IF(34), DAY2)
      WRITE (7, 18) DAY1, IF(30), (IF(I), I = 1, 24), (IF(I), I =
```

```
     X 26, 30), DAY2, IF(37)
C
C CHANGE FIRST '0' TO '1'
C
   18 FORMAT ('1', 40X, A4, I2, ', 19', I2, '.', /, '0DEAR ', I6, ':',
     X /, '0     RE:  ', 24A1, 4A4,I6,'.' / '0     THE BOOK LISTED ABOVE
     X WAS BORROWED BY YOU ON ', A4, I2, ', 19', I2, '.', /, '0SINCE THE
     X LOAN PERIOD FOR THIS BOOK IS ', I2, ' DAYS, IT IS NOW OVER-', /,
     X '0DUE.  PLEASE RETURN IT TO THE LIBRARY FROM WHICH YOU BORROWED I
     XT.', /, '0', 40X, 'CIRCULATION DEPARTMENT,',/,41X,'MAIN LIBRARY.')
      NOTICE = NOTICE + 1
      GO TO 10
20    WRITE (LIB,22) DAY1, NUMDEL, NOTICE, NOVER4
   22 FORMAT ('0', A4, I2, ', 19', I2, /, '0NUMBER OF RESERVES DELETED =
     X ', I4, /, ' NUMBER OF OVERDUE NOTICES PRINTED = ', I4, /,' NUMBER
     X OF BOOKS OVERDUE 4 WEEKS OR MORE = ', I4)
      IF (NOVER4.EQ.0) RETURN 1
      WRITE (LIB,23) ((MORE4(I,J), J = 1, 30), I = 1, NOVER4)
   23 FORMAT ('0THE FOLLOWING ARE 4 OR MORE WEEKS OVERDUE:', /, '+______
     X_____________________________________', /, '0', 8X, 'AUTHOR', 16X,
     X 'CATALOG NUMBER', 5X, 'USER ID', 3X, 'WEEKS OVERDUE', //,
     X (1X, 24A1, 5X, 4A4, 5X, I6, 8X, I3))
      RETURN 1
      END




      SUBROUTINE ENROLL(INPT,*)
C  MTS VERSION....LAST UPDATED ON OCT. 21, 1970 BY J. DIMSDALE
C
C THIS SUBROUTINE HANDLES THE COMMAND ENROLL.
C IT IS ALSO USED WITH THE OFF-LINE MODULE 'BATCHENROLL'.
C
      IMPLICIT INTEGER(A-W),LOGICAL(X-Z),REAL($)
      DATA KBLA/'    '/,ACCT/0/
      DATA STAR/'*   '/
      DATA QMARK/'?   '/
      DIMENSION NAMKEY(500,6),IDNKEY(500,2),ADDR(6),NAME(5),DEPT(2),
     CPHON(2),FSL(19),USER1(20),USER2(20),DEL(5),CON(5),NUMBER(9),
     CSTAT(4)
      LOGICAL FSFLAG, REJECT, ALFLAG
      EQUIVALENCE (NAME(1),USER1(1)),(STAT(1),USER1(6)),
     C(ADDR(1),USER1(10)),(PHON(1),USER1(16)),(DEPT(1),USER1(18)),
     C(STATUS,USER2(1)),(TITLE,USER2(2)),(IDNUM,USER2(3)),(J,USER2(4)),
     C(REJECT,USER2(5)),                    (NAMEIX,USER2(8))
      COMMON /TABLES/NAMKEY,IDNKEY,FSL,DEL,CON,FSL24(26)
      COMMON /PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL,USTOR1,CSTOR1,FSIN24,
     CALFLAG,MAXPTR,DUMMY(10),FLOOR
      COMMON XTRACE,INDEX,NDATE
C
C
```

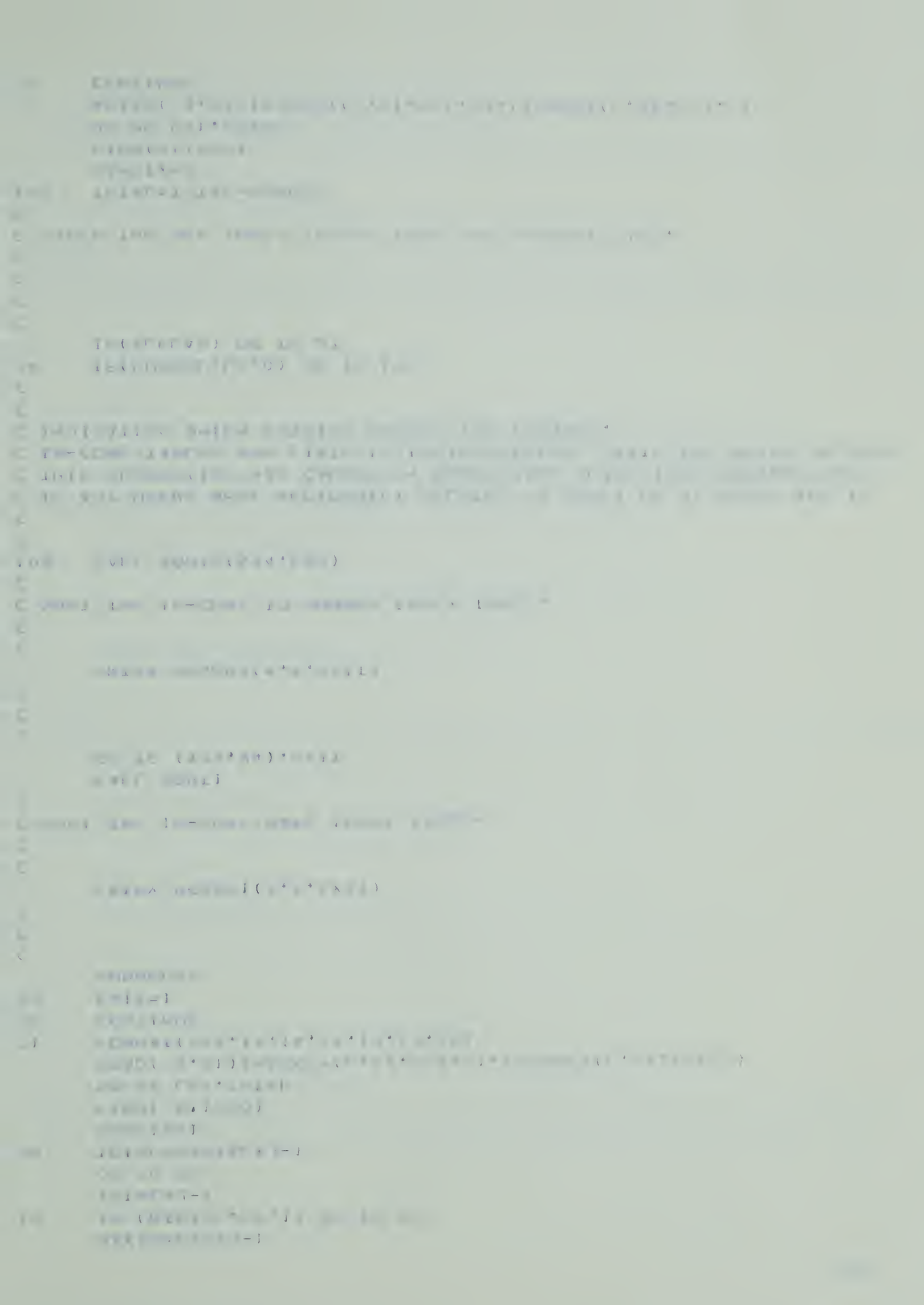


```
C
C
C
C RESET FLAG 'ALFLAG' TO .FALSE. TO INDICATE THAT THIS PROGRAM WAS NOT
C CALLED BY SUBROUTINE ALTER
C
      ALFLAG = .FALSE.
C
C INPT SPECIFIES THE FILE IN WHICH THE RAW ENTRIES FOR
C NEW USERS ARE TO BE FOUND.
C
C
      IF (INPT.EQ.5) WRITE(6,4)
4     FORMAT(///'  C) TYPE IN THE FOLLOWING INFORMATION ABOUT EACH NEW U
     CSER.'//5X,'EACH RESPONSE MUST BEGIN DIRECTLY UNDER THE FIRST LETTE
     CR OF THE REQUIRED ITEM.'//,
     C' SURNAME          INIT STATUS ADDRESS                    PHONE    DEPT
     C        IDNUMBER')
C
C
C SAVE THE VALUE OF 'TOTAL', THE CURRENT NUMBER OF RECORDS IN FILE
C 'NAMEKEY' AND, HENCE, THE NUMBER OF CURRENTLY ACTIVE USERS.
C
      OTOTAL = TOTAL
C
C
C RESET INDEX-TABLE POINTER
C
      J=1
C
C SAVE ORINGINAL VALUES OF USTOR1 AND FSINDX, THE PARAMETERS FOR FILE
C ''FREESTOR''.
C
      SAVE1=USTOR1
      SAVE2=FSINDX
C
C
C
C IF FREE STORAGE LIST IS NONEMPTY THEN SET FSFLAG=.FALSE. SO THAT
C VACANT SLOTS IN FILE MAINUSER WILL BE FILLED BEFORE THE FILE IS
C EXTENDED.  OTHERWISE, SET FSFLAG=.TRUE. AND LIBID (THE DIRECT ACCESS
C INDEX FOR FILE MAINUSER) = LAST RECORD +1 SO THAT THE INCOMING RECORDS
C WILL BE ADDED TO THE END OF THE FILE.
C
C
      IF( FSINDX.LE.0) GO TO 8
      FSFLAG=.FALSE.
      GO TO 9
8     FSFLAG = .TRUE.
      LIBID = MAXID + 1
C
C
C RESET FLAG 'REJECT'.  IF ANY ERRORS ARE DETECTED BY CHECK OR CHECKR
C DURING THE PROCESSING OF AN INCOMING USER RECORD THEN 'REJECT' IS SET
```

```
C TO .TRUE., MESSAGES INDICATING THE NATURE OF THE ERRORS ARE PRINTED AT
C THE TERMINAL, NO ENTRIES ARE MADE IN THE PERMANENT FILES, AND ANOTHER
C USER RECORD IS READ.
C
9     REJECT = .FALSE.
C
C
C
      GO TO 10
140   WRITE(6,141)
141   FORMAT('  C) IF YOU ARE FINISHED, TYPE *.')
C
C READ IN  AND PROCESS A NEW USER'S RECORD
C
10    READ(INPT,11,END=17) NAME,STAT,ADDR,PHON,DEPT,NUMBER
      WRITE(7,11) NAME,STAT,ADDR,PHON,DEPT,NUMBER
11    FORMAT(5A4,4A1,3X,6A4,A4,A3,1X,2A1,10X,9A1)
C
C
C TYPING A STAR INDICATES THAT YOU HAVE FINISHED ENTERING USER DATA.
      IF(NAME(1).EQ.STAR) GO TO 17
C
C DOES THE USER REQUIRE PROMPTING?
      IF(NAME(1).EQ.QMARK) GO TO 140
C
C CHECK INPUT FIELDS FOR ERRORS.
      CALL CHECK(6,NAME,1,5,DUMMY,.TRUE.,&72)
150   CALL CHECKR(6,STAT,1,2,STATUS,&74)
155   CALL CHECKR(6,STAT,3,4,TITLE,&76)
160   CALL CHECKR(6,NUMBER,1,9,IDNUM,.FALSE.,&78)
C
C DID WE FIND IN ANY ERRORS IN THE INPUT FIELDS?
      IF (REJECT) GO TO 70
C
C
C ENTER THE NEW USER'S NAME AND IDNUMBER INTO THE IN-CORE INDEX TABLES
C
C
      DO 180 K = 1,5
      NAMKEY(J,K) = NAME(K)
180   CONTINUE
      IDNKEY(J,1) = IDNUM
C
C
C IF AN EMPTY SLOT EXISTS IN THE MIDDLE OF THE MAINUSER FILE THEN STORE
C THE DATA ABOUT THE NEW USER  IN THE EMPTY SLOT AND UPDATE THE FREE
C STORAGE LIST.  OTHERWISE, ADD THE NEW DATA TO THE END OF THE FILE.
C
C
      IF (FSFLAG) GO TO 15
12    IF(FSINDX.GT.0) GO TO 14
      IF(USTOR1.LE.1) GO TO 13
      USTOR1=USTOR1-1
      USTOR=USTOR1
```

```
      FIND(9'1000*USTOR)
      READ( 9,56) FSINDX,FSL
      GO TO 12
C
C IF ALL VACANT SLOTS HAVE BEEN FILLED THEN SET FSFLAG=.TRUE. AND
C SET THE DIRECTACCESS INDEX FOR FILE MAINUSER= LAST RECORD+1 SO THAT
C THE INCOMING USER RECORD AND ALL SUBSEQUENT ONES WILL BE ADDED TO
C THE END OF FILE MAINUSER.
C
13    FSFLAG=.TRUE.
      LIBID=MAXID+1
      GO TO 15
C
14    LIBID=FSL(FSINDX)
      FSINDX=FSINDX-1
C
C
C ENTER THE LOCATION OF THE NEW USER'S RECORD IN THE NAME AND ID NUMBER
C TABLES.
C
15    NAMKEY(J,6)=LIBID
      IDNKEY(J,2)=LIBID
C
C ENTER THE  NEW USER'S RECORD INTO THE MAINUSER FILE.
C
      FIND( 4'1000*LIBID)
      WRITE( 4,16)NAME,STATUS,TITLE,ADDR,PHON,DEPT,
     CIDNUM,ACCT,NDATE,LIBID
16    FORMAT(5A4,2I1,7A4,A3,2A1,I9,I1,I4,7X,I4)
      IF(XTRACE)WRITE(6,101) NAME,STATUS,TITLE,ADDR,PHON,
     CDEPT,(IDNKEY(J,K),K=1,2)
101   FORMAT(1X,5A4,1X,I1,1X,I1,3X,7A4,A3,1X,2A1,10X,I9,1X,I4)
      J=J+1
      LIBID = LIBID + 1
      GO TO 10
C
C
C IF ENROLLMENT TERMINATED AFTER FAILURE TO ENROLL FIRST NEW USER THEN
C RETURN WITHOUT MAKING ANY CHANGES IN THE IN-CORE TABLES OR IN THE
C FILES
C
17    IF(J.LE.1)  RETURN 1
C
C
C IF THE INDEX TABLES EXISTED PREVIOUSLY THEN APPEND THEM TO THE IN-CORE
C TABLES WHICH  CONTAIN THE NEW USERS.
C
18    IF (MAXID.GE.LIBID) GO TO 19
```

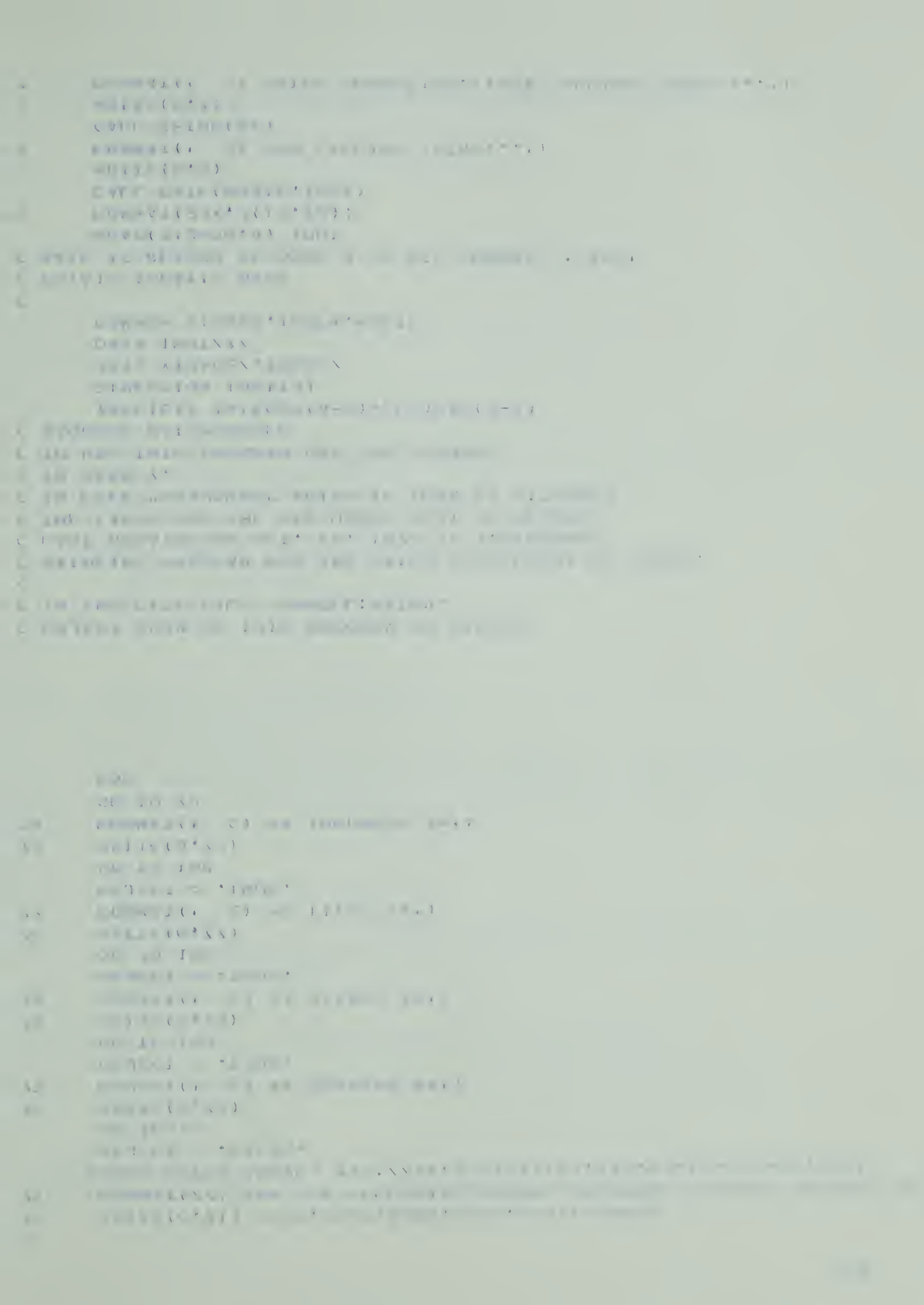
```
      MAXID=LIBID-1
19    IF (OTOTAL.GE.1) GO TO 20
      TOTAL=J-1
      GO TO 23
20    TOTAL=OTOTAL+J-1
      NAMEIX=1
      FIND( 8'1000)
      DO 22 L=J,TOTAL
      READ( 8,21)(NAMKEY(L,K),K=1,6),(IDNKEY(L,K),K=1,2)
21    FORMAT(5A4,1X,I4,1X,I9,1X,I4)
22    CONTINUE
23    EXIT=1
      NUMDEL=0
C
C
C
      ENTRY RESOR1(*,*,EXIT)
C
C
C SORT THE IN-CORE NAME INDEX TABLE.
C
      CALL SORT1
      GO TO (103,24),EXIT
C
C
C
      ENTRY RESOR2(*,*,EXIT)
C
C
C SORT THE IN-CORE ID NUMBER INDEX TABLE.
C
103   CALL SORT2(&24,&27)
C
C
C IF ANY USERS WERE ACCIDENTLY DELETED BY SORT1 OR BY SORT2 AND IF
C THIS SUBROUTINE WAS CALLED BY SUBROUTINE ALTER THEN RESTORE THE
C IN-CORE TABLES AND LISTS TO THEIR ORIGINAL STATE AND PRINT MESSAGE
C INDICATING WHICH ENTRIES CAUSED THE TROUBLE.
C
C
24    IF(NUMDEL.LE.0) GO TO 105
      IF(ALFLAG) GO TO 27
C
C
C
C
C ENTER THE NEW INDEX TABLES INTO THE NAMEKEY FILE.
C
105   TOTAL=TOTAL-NUMDEL
      NAMEIX=1
      FIND(8'1000)
      DO 25 L=1,TOTAL
      WRITE( 8,21)(NAMKEY(L,K),K=1,6),(IDNKEY(L,K),K=1,2)
25    CONTINUE
```



```
C
C
C ENTER THE UPDATED FILE PARAMETERS INTO THE FIRST RECORD OF THE MAIN-
C USER FILE.
C
      LIBID=1
      FIND(4'1000)
      WRITE( 4,56) TOTAL,MAXID,USTOR1,OSTOR1,MAXPTR,FLOOR
C
C UPDATE THE LAST NONEMPTY RECORD IN FILE FREESTOR
C
      USTOR=USTOR1
      FIND(9'1000*USTOR)
      WRITE( 9,56) FSINDX,FSL
C     CALL LOGDSK             ***** REMOVED FOR MTS VERSION *******
      RETURN 1
C
C
C
27    WRITE(6,28)
28    FORMAT(//' *** DISASTER     THIS UPDATE IS BEING SCRAPPED    ***'
     C//' THE USER FILES AND TABLES REMAIN IN THEIR ORIGINAL STATE'//,
     C' PLEASE CHECK THE NAMES AND IDNUMBERS IN THE FOLLOWING PAIRS',
     C'  OF ENTRIES')
      WRITE(6,550) NUMDEL,DEL, CON
550   FORMAT('  C) ***** NUMDEL = ',I2,' DEL = ',5I4,' CON = ',5I4)
C
C
      DO 30 K=1,NUMDEL
      LIBID=DEL(K)
      FIND(4'1000*LIBID)
      READ( 4,16) USER1
      LIBID=CON(K)
      FIND(4'1000*LIBID)
      READ( 4,16) USER2
      WRITE(6,29) USER1,USER2
29    FORMAT(//2(/1X,5A4,1X,I1,1X,I1,4X,7A4,A3,1X,2A1,9X,I9,1X,I1,1X,I4)
     C)
30    CONTINUE
C
C
C RESTORE THE IN-CORE NAME AND ID NUMBER TABLES TO THEIR ORIGINAL STATE.
C
C
      CALL SETUP(&33)
33    IF(ALFLAG) RETURN 2
      RETURN 1
C
C
56    FORMAT(20I4)
C
C
C MESSAGES FOR ERRORS ENCOUNTERED DURING PROCESSING OF A NEW USER'S
C RECORD
```

```
C
70     WRITE(6,71) NAME,STAT,ADDR,PHON,DEPT,NUMBER
71     FORMAT(//' *** THE FOLLOWING ENTRY HAS BEEN REJECTED BECAUSE OF ER
      CRORS NOTED ABOVE. ***'//1X,5A4,4A1,4X,7A4,A3,1X,2A1,9X,9A1)
       REJECT = .FALSE.
       GO TO 10
72     WRITE(6,73)
73     FORMAT('  C) ** SURNAME **')
       REJECT = .TRUE.
       GO TO 150
74     WRITE(6,75)
75     FORMAT('  C) ** STATUS **')
       REJECT = .TRUE.
       GO TO 155
76     WRITE(6,77)
77     FORMAT('  C) ** TITLE **')
       REJECT = .TRUE.
       GO TO 160
78     WRITE(6,79)
79     FORMAT('  C) ** IDNUMBER **')
       GO TO 70
       END
```

```
C OBJECT FORM OF THIS PROGRAM IS STORED
C IN THE DISK FILE "ENROLL:MAIN".
C
C MAINLINE PROGRAM FOR THE BATCH ENROLLMENT OF USERS.
C LAST UPDATED ON OCT. 28, 1970 BY J.DIMSDALE
C THE CARDS FOR THE NEW USERS MUST BE PLACED
C IN FILE "NEWUSERS" WHICH IN TURN IS ATTACHED
C TO DSRN 7.
C TO USE THIS PROGRAM USE THE COMMAND
C $SOURCE BATCHENROLL
       IMPLICIT INTEGER(A-W),LOGICAL(X-Z)
       DIMENSION IOUT(3)
       DATA XTRACE/.TRUE./
       DATA INPT/7/
       COMMON XTRACE,INDEX,NDATE
C
C OBTAIN TODAY'S DATE
C DATE IS PLACED IN DSRN 3 BY MTS COMMAND 'TIME'
       READ(3'3000,4) IOUT
4      FORMAT(23X,3(1X,I2))
       CALL DATE(NDATE,IOUT)
       WRITE(6,5)
5      FORMAT('  C) NOW CALLING SETUP...')
       CALL SETUP(&6)
6      WRITE(6,7)
7      FORMAT('  C) SETUP COMPLETED...NOW CALLING ENROLL...')
```



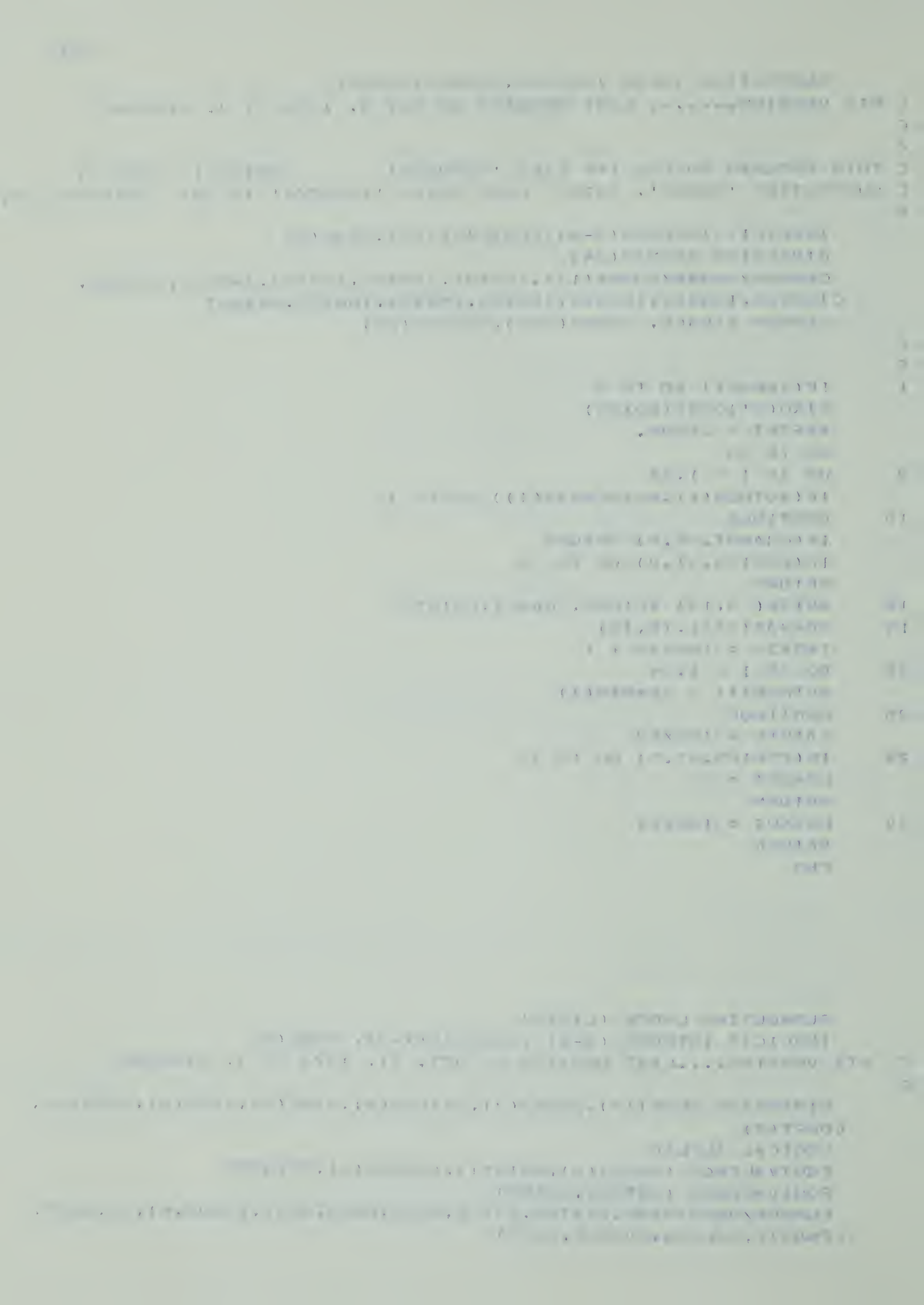
```
      CALL ENROLL(INPT,&9)
9     WRITE(6,10)
10    FORMAT('  C) BATCH ENROLLMENT OF USERS COMPLETED.')
      STOP
      END



      SUBROUTINE ENQUIR (*)
C MTS VERSION....LAST UPDATED ON OCT. 22, 1970 BY J. DIMSDALE
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      DIMENSION AUTHOR(24),CATLOG(4),FILCAT(4),RLIST(3)
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      COMMON/PARAM/DUMMY(10),INDX01,INDX02,INDX03,INDX04,INDX08,
     CINDX20,INDX21,INDX22,INDX23,INDX24
      EQUIVALENCE (AUTHOR(1),INPUT(3)),
     C           (CATLOG(1),INPUT(27)),
     C           (LIBID,INPUT(31)),
     C           (NUMSUR,INPUT(32)),
     C           (FILCAT(1),IF(26)),
     C           (RLIST(1),IF(31))
C
C
      MFILE = 1
C
C INDX01 HAS BEEN SET BY SUBROUTINE QUICK.
      FIND (MFILE'1000*INDX01)
   20 IX = 1
C
C READ RECORD FROM LOANS FILE.
   21 READ (MFILE, 22,END = 65) (IF(I), I = 1, 36)
   22 FORMAT (24A1, I2, 4A4, 4I6, I4, I3, I2)
C
C  COMPARE AUTHOR SURNAMES
      DO 25 ID = 1, NUMSUR
      IF(IF(ID)-AUTHOR(ID))   20,25,65
   25 CONTINUE
C
C  COMPARE CATALOG NUMBERS
      DO 30 IE = 1, 4
      IF(FILCAT(IE).NE.CATLOG(IE))   GO TO 20
   30 CONTINUE
C
C  HAVE FOUND FIRST COPY;  CHECK THIS FOR ID NUMBER
      DO 35 IG = 1, 3
      IF(RLIST(IG).EQ.LIBID)   GO TO 50
```

```
   35 CONTINUE
C IF NOT CORRECT ID NUMBER, THEN TRY NEXT COPY OF BOOK
      IX = IX + 1
C
C LAST COPY?
      IF (IX.LE.IF(36)) GO TO 21
C
C IF ALL COPIES EXAMINED THEN THIS USER DOES NOT
C HAVE A RESERVATION ON THIS BOOK.
      WRITE (NOUT, 37) (INPUT(I), I = 3, 31)
   37 FORMAT ('  C) ',24A1, ' AUTHOR', /, 5X, 4A4, 9X, 'CATALOG NUMBER',
     X /, 5X, I6, 19X, 'YOUR ID NUMBER', /, 5X, 'NO RECORD OF THIS RESER
     XVATION.')
      RETURN 1
C
C HAVE FOUND USER'S RESERVATION. IS IT AVAILABLE?
   50 IF (IF(30).NE.0.OR.IF(31).NE.INPUT(31)) GO TO 60
      WRITE (NOUT, 52)
   52 FORMAT ('  C) COLLECT FROM LIBRARY WITHIN THIRTY MINUTES.' /
     X '      (USE SEARCH COMMAND TO FIND LIBRARY NAME)')
      RETURN 1
   60 WRITE (NOUT, 61)
   61 FORMAT ('  C) NOT YET AVAILABLE')
   63 CONTINUE
      RETURN 1
   65 WRITE (NOUT, 66)
66    FORMAT('  C) USE MAIN AUTHOR.')
67    RETURN 1
      END



      SUBROUTINE IDENT(LIBID,SEE,*)
C
C THIS SUBROUTINE PROCESSES THE USER'S IDENTIFICATION,
C GETS THE USER'S RECORD, AND DETERMINES THE USER'S
C CURRENT BORROWING STATUS.
C
C   MTS VERSION......LAST UPDATED ON SEPT. 30, 1970 BY J. DIMSDALE
C
      IMPLICIT INTEGER(A-W),LOGICAL(X-Z),REAL($)
      LOGICAL*1 TNAME(20),TLINE(80)
      DIMENSION NAME(5),NUMBER(10),LINE(80),LINE1(80),BUFFR1(20),
     CBUFFR2(20)
      COMMON/USER/DUMMY(20),ACCT
      COMMON XTRACE,INDEX,NDATE,IF(844),LAST(20),INPUT(36),LINK(34),
     CLOAN(20,40),INDX26
      COMMON/COMAND/LIST(22),KBLA,KYES
      EQUIVALENCE(IF(600),LINE(1)),(IF(680),LINE1(1)),
     C(NUMBER(1),LINE(1)), (NAME(1),TNAME(1)),(LINE1(1),TLINE(1))
      DATA K0/'0   '/
```

```
      DATA COMMA/',    '/,PERIOD/'.    '/
      DATA QMARK/'?    '/
      DATA SCORE/'_    '/
C
1     FILE=6
      GO TO 99
95    WRITE(6,96)
96    FORMAT('  C) ENTER USER''S IDENTIFICATION.')
99    READ(5,100,END=95) LINE
C UPDATE THE SYSTEM LOG
      WRITE(7,100) LINE
100   FORMAT(80A1)
C
C DOES THE USER REQUIRE PROMPTING?
      IF( LINE(1) .EQ.QMARK) GO TO 95
C
C
C
      ENTRY IDENT2(LIBID,SEE,*)
C
C
      DO 101 I = 1,80
      IF( LINE(I).NE.KBLA) GO TO 103
101   CONTINUE
      RETURN 1
103   IF( LINE(I).LT.K0) GO TO 110
      II = I + 10
      CALL CHECKR (6,LINE,I,II,IDNUM,&99)
      IF(.NOT.XTRACE) WRITE(6,90) IDNUM
90    FORMAT(1X,I10)
      IF(IDNUM.GT.10000) CALL LOOK2(IDNUM,LIBID,SEE,&25,&33)
      LIBID = IDNUM
      CALL LOOK3(LIBID,SEE,6,&25,&33)
110   DO 111 J = 1,17
C PULL OUT THE USER'S SURNAME
      IF(LINE(I).EQ.KBLA.AND.LINE(I+1).EQ.KBLA) GO TO 115
      IF(LINE(I).EQ.COMMA)   GO TO 115
      IF(LINE(I).EQ.KBLA) LINE(I) = SCORE
210   LINE1(J) = LINE(I)
      I = I + 1
111   CONTINUE
      WRITE(6,112)
112   FORMAT('  C) NO MORE THAN 16 LETTERS ALLOWED IN SURNAME.')
      GO TO 99
115   IF(J.GT.16)  GO TO 120
      DO 116 K = J,16
      LINE1(K) = KBLA
116   CONTINUE
120   J = 16
      DO 123 K = 1 , 15
      I = I + 1
      IF (I.GT.80)   GO TO 124
      IF(LINE(I).EQ.KBLA)   GO TO 123
      IF(LINE(I).EQ.PERIOD)  GO TO 123
```

```
      J = J + 1
      IF(J.LE.20) GO TO 122
      WRITE(6,121)
121   FORMAT('  C) NO MORE THAN 4 INITIALS ALLOWED.')
      GO TO 99
122   LINE1(J) = LINE (I)
123   CONTINUE
124   J = J + 1
      IF(J.GT.20) GO TO 126
      DO 125 K = J,20
125   LINE1(K) = KBLA
C COMPRESS THE NAME OF THE USER
126   DO 127 K =1, 20
127   TNAME(K) = TLINE(4*K-3)
      IF(.NOT.XTRACE) WRITE(6,128) NAME
128   FORMAT(20A4)
      CALL LOOK1(NAME,LIBID,SEE,&25,&33)
C
C
25    IF(INPUT(1).NE.LIST(5)) RETURN
      GO TO(26,30,40,50), ACCT
26    IF(XTRACE) RETURN
28    WRITE(6,29) LIBID
29    FORMAT(' LIBRARY IDNUMBER IS  ',I4)
      RETURN
C
C
30    WRITE(6,31)
31    FORMAT('  C) THIS USER HAS BOOKS OVERDUE        TYPE YES IF YOU WAN
     CT THIS TRANSACTION TO BE COMPLETED')
      READ(5,32,END=33) ANS
32    FORMAT(A4)
C UPDATE THE SYSTEM LOG
      WRITE(7,32) ANS
      IF(ANS.EQ.KYES) GO TO 26
33    RETURN 1
C
C
40    WRITE(6,41)
41    FORMAT('  C) *** FOR SHAME       THIS USER DOES NOT RETURN THE BOOK
     CS THAT HE BORROWS          *****'/' *** THEREFORE, THIS TRANSACTION
     CIS BEING TERMINATED ')
      GO TO 33
C
C
50    WRITE(6,51) LIBID
51    FORMAT('  C) *** USER ACCOUNT NO. ',I5,' IS INACTIVE. TRANSACTION
     C TERMINATED.  ****')
      GO TO 33
C
      END
```

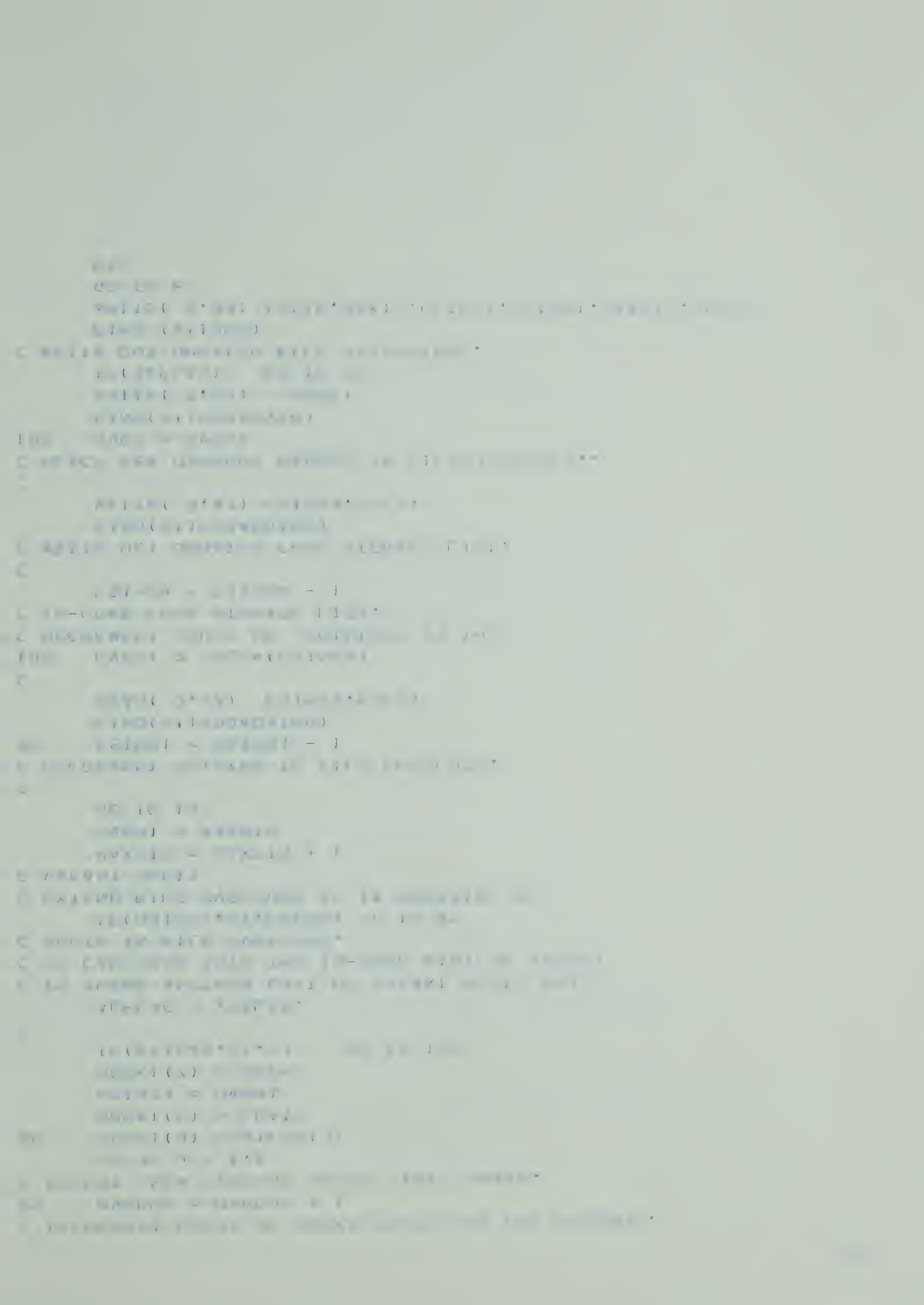
```
      SUBROUTINE INDXR (NEWNAM,COPIES,XRESET)
C MTS VERSION..... LAST UPDATED ON MAY 8, 1970 BY J. DIMSDALE
C
C
C THIS PROGRAM BUILDS THE FILE 'AUTHORS'          WHICH IS USED BY
C SUBROUTINE 'QUICK'. NOTE:  LAST ENTRY 'ZZZZZZZ' IS NOT  WRITTEN OUT.
C
      IMPLICIT INTEGER(A-W),LOGICAL(X-Z),REAL($)
      DIMENSION NEWNAM(24)
      COMMON/PARAM/DUMMY(10),INDX01,INDX02,INDX03,INDX04,INDX08,
     CINDX20,INDX21,INDX22,INDX23,INDX24,INDX25,MAXAUT
      COMMON XTRACE, DUMBY(868),AUTHOR(24)
C
C
1     IF(XRESET) GO TO 9
      FIND(0'1000*INDX25)
      XRESET = .TRUE.
      GO TO 20
9     DO 10 I = 1,24
      IF(AUTHOR(I).NE.NEWNAM(I)) GO TO 18
10    CONTINUE
      IF(LOANPT.NE.0) RETURN
      IF(COPIES.GT.0) GO TO 30
      RETURN
18    WRITE( 0,19) AUTHOR,LOANPT,CATPTR
19    FORMAT(24A1,I5,I6)
      INDX25 = INDX25 + 1
20    DO 25 I = 1,24
      AUTHOR(I) = NEWNAM(I)
25    CONTINUE
      CATPTR = INDX03
29    IF(COPIES.GT.0) GO TO 30
      LOANPT = 0
      RETURN
30    LOANPT = INDX01
      RETURN
      END



      SUBROUTINE LBOOK (LIBID)
      IMPLICIT INTEGER (A-W) ,LOGICAL(X-Z), REAL($)
C  MTS VERSION....LAST UPDATED ON OCT. 21, 1970 BY J. DIMSDALE
C
      DIMENSION BOOK1(7),BOOK2(7),CATLOG(4),NAME(5),ADDR(6),PHON(2),
     CDEPT(2)
      LOGICAL ALFLAG
      EQUIVALENCE (BOOK1(6),POINT1),(BOOK2(6),POINT2)
      EQUIVALENCE (OSTOR1,OSTOR)
      COMMON/USER/NAME,STATUS,TITLE,ADDR,PHON,DEPT,IDNUM,K1,K2,ACCT,
     CENDATE,ONLOAN,OVRDUE,OVER1
```



```
      COMMON/TABLES/NAMKEY(500,6),IDNKEY(500,2),FSL(19),DEL(5),CON(5),
     CFSL24(26)
      COMMON/PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL,USTOR1,OSTOR1,FSIN24,
     CALFLAG,MAXPTR,DUMMY(10),FLOOR
C
C THIS SECTION HANDLES LOAN TRANSACTIONS.
C
C OBTAIN USER'S RECORD.
      CALL LOOK3( LIBID,0,6,&5,&5)
C INCREMENT COUNT OF BOOKS ON LOAN TO USER.
5     ONLOAN = ONLOAN + 1
C IF NECESSARY CHANGE STATUS OF USER'S ACCOUNT
      IF(ACCT.EQ.0)    ACCT = 1
C PUT UPDATED USER RECORD BACK INTO FILE USERS.
8     CALL PUT(&10)
10    RETURN
C
C
      ENTRY RBOOK (LIBID,CATLOG)
C
C THIS SECTION HANDLES RETURN TRANSACTIONS.
C
C OBTAIN USER'S RECORD.
      CALL LOOK3 (LIBID,0,6,&20,&20)
C DECREMENT COUNT OF BOOKS ON LOAN TO USER.
20    ONLOAN = ONLOAN - 1
C IF NECESSARY CHANGE STATUS OF USER'S ACCOUNT.
      IF(ONLOAN.EQ.0)       ACCT=0
C DOES THE USER HAVE ANY BOOKS OVERDUE?
      IF (OVRDUE.LE.0)  GO TO 8
C IF YES THEN SEARCH THROUGH HIS CHAIN OF
C OVERDUE BOOKS TO SEE IF BOOK BEING  RETURNED
C IS ONE OF THE BOOKS ON THAT CHAIN.
      I = 0
      POINT1 = OVER1
C END OF CHAIN OF OVERDUES?
25    IF (POINT1.LE.0)  GO TO 8
C
      I = I + 1
      OVER = POINT1
      POINT3 = POINT2
      DO 30 J = 1, 7
      BOOK2(J) = BOOK1(J)
30    CONTINUE
      FIND(0'1000*OVER)
      READ ( 0,32)  BOOK1
32    FORMAT(4A4,I4,I3,I2)
      DO 35 J = 1, 4
      IF(BOOK1(J).NE.CATLOG(J))   GO TO 25
35    CONTINUE
C DECREMENT COUNT OF BOOKS THAT USER HAS OVERDUE.
      OVRDUE = OVRDUE - 1
C IF NECESSARY CHANGE STATUS OF USER'S ACCOUNT.
      IF(ONLOAN.GT.0.AND.OVRDUE.EQ.0)       ACCT = 1
```

```
      IF(FSIN24.LT.26)   GO TO 40
      FSIN24 = 0
      OSTOR1 = OSTOR1 + 1
C
C WRITE OUT UPDATED FILE PARAMETERS.
      FIND (4'1000)
      WRITE( 4,39) TOTAL,MAXID,USTOR1,OSTOR1,MAXPTR,FLOOR
39    FORMAT(9I4)
C
C ADD POINTER TO THIS OVERDUE RECORD TO THE
C LIST OF FREE STORAGE LOCATIONS IN FILE OVERDUES.
40    FSIN24 = FSIN24 + 1
      FSL24(FSIN24) = POINT2
C WRITE OUT UPDATED FREE STORAGE LIST.
      FIND(9'1000*OSTOR)
      WRITE( 9,42) FSIN24,FSL24
42    FORMAT(I2,26I3)
      IF (I.GE.2)   GO TO 45
      OVER1 = POINT1
      GO TO 8
45    OVER = POINT3
      POINT2 = POINT1
      FIND(0'1000*OVER)
      WRITE( 0,32)   BOOK2
      GO TO 8
C
C
C
      ENTRY OBOOK (LIBID,CATLOG,LDATE,WEEKS)
C
C THIS SECTION ADDS OVERDUE BOOKS TO A USER'S
C CHAIN OF OVERDUES.
C
      ALFLAG = .TRUE.
C OBTAIN USER'S RECORD.
      CALL LOOK3(LIBID,0,6,&60,&60)
C CHANGE STATUS OF USER'S ACCOUNT.
60    ACCT = 2
      IF(WEEKS.GE.4.AND.STATUS.GT.5) ACCT = 3
      IF (OVRDUE.LT.1)   GO TO 85
      I = 0
      POINT1 = OVER1
C END OF CHAIN OF OVERDUES?
65    IF (POINT1 .LE. 0 )  GO TO 85
C
      I = I + 1
      OVER = POINT1
      FIND(0'1000*OVER)
      READ ( 0,32)  BOOK1
      DO 75 J = 1,4
      IF(BOCK1(J).NE.CATLOG(J))   GO TO 65
75    CONTINUE
      GO TO 8
C
```

```
C INCREMENT COUNT OF BOOKS THAT USER HAS OVERDUE.
85     OVRDUE = OVRDUE + 1
C INSERT  NEW OVERDUE RECORD INTO CHAIN.
       DO 90 J = 1,4
90     BOOK1(J) =CATLOG(J)
       BOOK1(5) = LDATE
       POINT1 = OVER1
       BOOK1(7) = WEEKS
       IF(FSIN24.GT.0)    GO TO 100
C
       ALFLAG = .FALSE.
C IS THERE ANOTHER LIST OF VACANT SLOTS THAT
C WE CAN READ INTO THE IN-CORE LIST OF VACANT
C SLOTS IN FILE OVERDUES.
       IF(OSTOR1.GT.FLOOR) GO TO 95
C EXTEND FILE OVERDUES IF IT CONTAINS NO
C VACANT SLOTS
       MAXPTR = MAXPTR + 1
       OVER1 = MAXPTR
       GO TO 105
C
C DECREMENT POINTER TO FILE FREELOCS.
95     OSTOR1 = OSTOR1 - 1
       FIND(9'1000*OSTOR)
       READ( 9,42)  FSIN24,FSL24
C
100    OVER1 = FSL24(FSIN24)
C DECREMENT COUNT OF  POINTERS IN THE
C IN-CORE FREE STORAGE LIST.
       FSIN24 = FSIN24 - 1
C
C WRITE OUT UPDATED FREE STORAGE LIST.
       FIND(9'1000*OSTOR)
       WRITE( 9,42) FSIN24,FSL24
C
C PLACE NEW OVERDUE RECORD IN FILE OVERDUES..
105    OVER = OVER1
       FIND(0'1000*OVER)
       WRITE( 0,32)   BOOK1
       IF(ALFLAG)   GO TO 8
C WRITE OUT UPDATED FILE PARAMETERS.
       FIND (4'1000)
       WRITE( 4,39) TOTAL,MAXID,USTOR1,OSTOR1,MAXPTR,FLOOR
       GO TO 8
       END
```

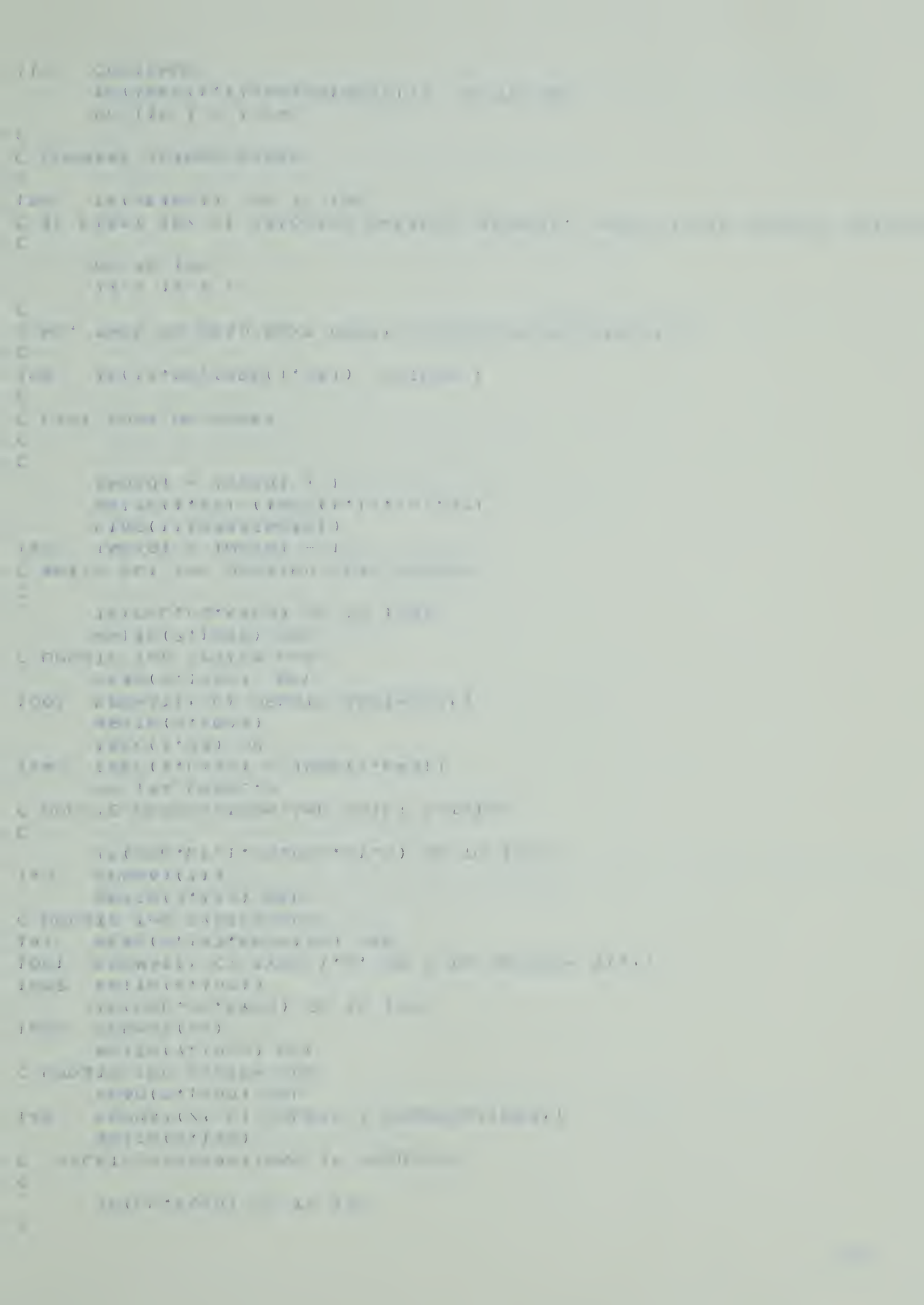


```
      SUBROUTINE LIST1(*)
C MTS VERSION.... LAST UPDATED ON  OCT. 4, 1970 BY J. DIMSDALE
C
C THIS SUBROUTINE HANDLES THE COMMANDS LIST, AVAILABLE,
C AND FIND.
C LIST IS PERFORMED BY ENTRY LIST2
C AVAILABLE IS PERFORMED BY ENTRY AVAIL.
C FIND IS PERFORMED BY ENTRY FINDIT.
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      COMMON/USER/NAME(5),DUMMY(20),LIBID
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), LAST(20), INPUT(36), LINK(34), IREC(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      COMMON/PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL,USTOR1,CSTOR1,FSIN24,
     CALFLAG,MAXPTR,INDX01,INDX02,
     CINDX03,INDX04,INDX08,INDX20,INDX21,INDX22,INDX23,INDX24
      DATA KX/'XXXX'/
      DIMENSION PRINT(25,33),AUTHOR(24),CATLOG(4),IOUT(3),RECCAT(4)
      EQUIVALENCE (PRINT(1,1),IF(1)),(AUTHOR(1),INPUT(3)),
     C            (CATLOG(1),INPUT(27)),(RECCAT(1),IREC(1,26))
      SEE = 0
      CALL IDENT(LIBID,SEE,&40)
C
C
      ENTRY LIST2(*)
C
C
      IF(LIBID.EQ.0) STOP 9
C OBTAIN THE DATE
      CALL CAY(NDATE,IOUT)
      WRITE(6,1) NAME,IOUT
1     FORMAT(///'  LIST OF BOOKS ON LOAN TO     ',5A4,10X,A4,I2,', 19',
     2I2//' AUTHOR',25X,'CATALOG NUMBER',5X,'OVERDUE',5X,'REQUESTED')
      INDEX = 0
      MFILE = 1
C SET POINTER TO START OF FILE 'LOANS'.
      FIND (MFILE'1000)
C READ A RECORD FROM THE FILE 'LOANS'.
25    READ(MFILE,26,END=40)(IREC(1,I),I=1,37)
26    FORMAT(24A1,I2,4A4,4I6,I4,I3,2I2)
      IF(.NOT.XTRACE) WRITE(6,27) (IREC(1,I),I=1,37)
27    FORMAT(' ** ',24A1,I2,4A4,4I6,I4,I3,2I2)
C IS THE BOOK OUT ON LOAN TO THE USER IN QUESTION?
      IF(IREC(1,30).NE.LIBID) GO TO 25
C IS BOOK ON RESERVE FOR ANOTHER USER?
      RESERV = KBLA
      IF(IREC(1,31).NE.0) RESERV = KX
C IS THE BOOK OVERDUE?
      OVER = KBLA
      IF(NDATE.GT.(IREC(1,37)+IREC(1,34))) OVER = KX
      WRITE(6,30)(IREC(1,I),I=1,24),(IREC(1,I),I=26,29),OVER,RESERV
```

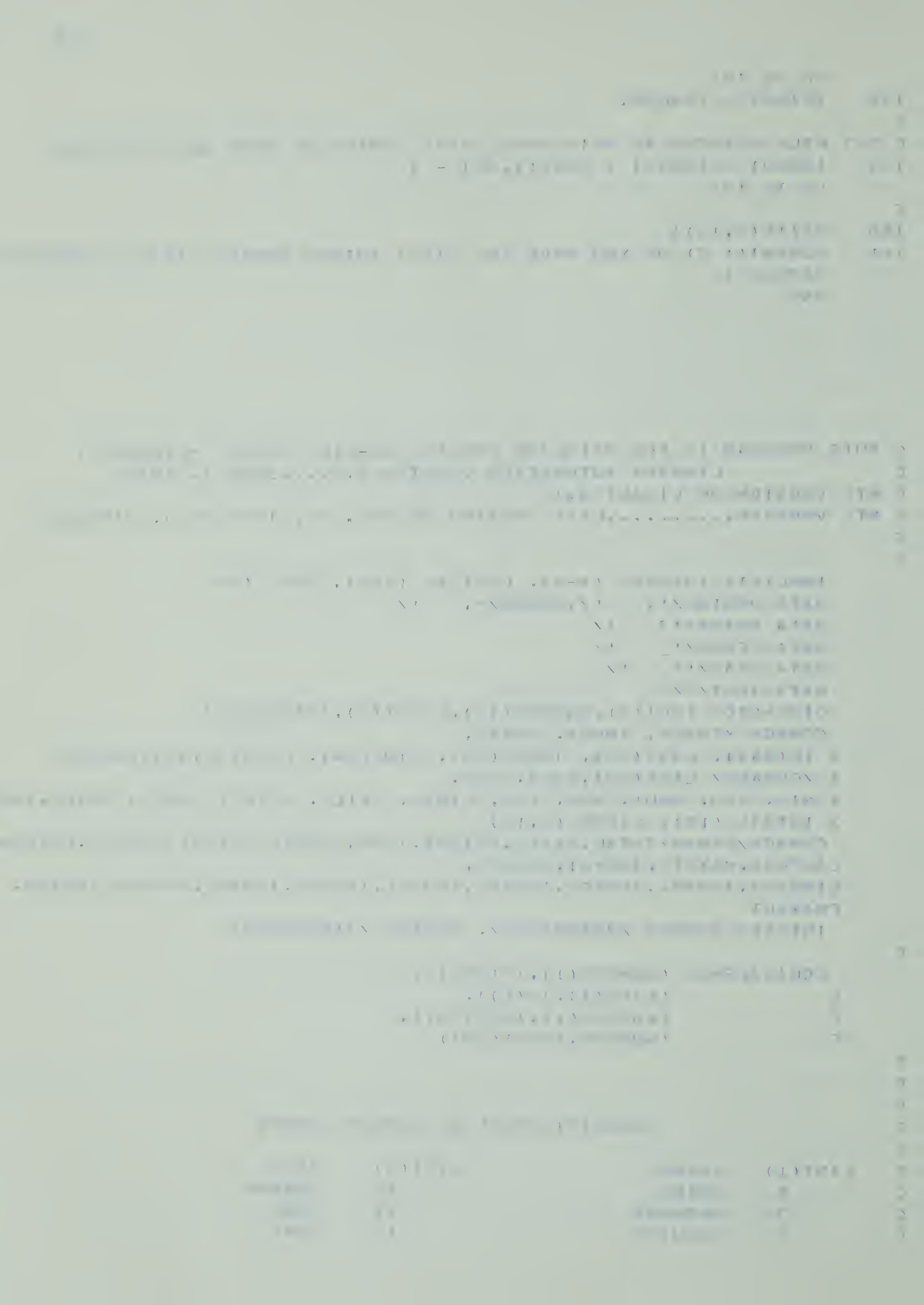
```
30     FORMAT(/1X,24A1,7X,4A4,3X,A4,8X,A4)
       INDEX = INDEX + 1
       GO TO 25
40     IF(INDEX.GT.0)  RETURN 1
43     WRITE(6,44)
44     FORMAT(11X,'***   NONE   ***')
       RETURN 1
C
C
       ENTRY AVAIL(*)
C
C
       SEE = 0
       FILE = 6
C OBTAIN DATE
       CALL DAY(NDATE,IOUT)
       MFILE=1
C SET POINTER TO START OF FILE 'LOANS'.
       FIND (MFILE'1000)
59     XFULL = .FALSE.
       INDEX = 0
C READ A RECORD FROM THE FILE 'LOANS'.
60     READ(MFILE,26,END=80) (IREC(1,I),I=1,31)
C IS THE BOOK STILL OUT?
       IF(IREC(1,30).NE.0)  GO TO 60
C IF IT IS NOT OUT, THEN DOES IT HAVE A RESERVATION ON IT?
       IF(IREC(1,31).EQ.0)  GO TO 60
C LOOKUP USER'S RECORD IN FILE 'USERS'.
       CALL LOOK3(IREC(1,31),SEE,FILE,&70,&60)
70     INDEX = INDEX + 1
       DO 71 I = 1,5
       PRINT(INDEX,I) = NAME(I)
71     CONTINUE
       DO 72 I = 1,24
72     PRINT(INDEX,I+5) = IREC(1,I)
       DO 73 I = 1,4
73     PRINT(INDEX,I+29) = IREC(1,I+25)
       IF(INDEX.LT.25) GO TO 60
       XFULL = .TRUE.
80     DO 92 K = 1,2
       WRITE(6,83) IOUT
83     FORMAT(     '  REQUESTED BOOKS WHICH ARE NOW AVAILABLE',25X,A4,I2,'
      C, 19',I2//' USER',20X,'AUTHOR',24X,'CATALOG NUMBER')
       IF(INDEX.EQ.0)   GO TO 43
       DO 90 J = 1,INDEX
       WRITE(6,85)  (PRINT(J,I),I=1,33)
85     FORMAT(/1X,5A4,4X,24A1,6X,4A4)
90     CONTINUE
       WRITE(6,91)
91     FORMAT('-'/'-')
92     CONTINUE
       IF(XFULL) GO TO 59
       RETURN 1
C
```

```
C
      ENTRY FINDIT(*)
C
C
      WRITE(6,99) AUTHOR,CATLOG
99    FORMAT(/' AUTHOR:  ',24A1,7X,'CATALOG NUMBER:  ',4A4)
      MFILE = 1
      SEE = 0
      FILE =6
      XFIRST = .TRUE.
100   IX = 1
C
C READ A RECORD FROM THE LOAN FILE
C
105   FIND (MFILE'1000*INDX01)
      READ(1,26,END=160)(IREC(1,I),I=1,37)
      IF(.NOT.XTRACE) WRITE(6,107) INDX01,(IREC(1,I),I=1,24),(IREC(1,I),
     XI=26,29)
 107  FORMAT('  ** ',24A1,5X,4A4)
      INDX01 = INDX01 + 1
C
C COMPARE CATALOG NUMBERS
C
      DO 120 I=1,4
      IF(CATLOG(I).NE.IREC(1,I+25)) GO TO 155
120   CONTINUE
C
C IS THE BOOK IN THE LIBRARY
C
      IF(IREC(1,30).EQ.0)  GO TO 132
C
C NO, THEN LOOKUP THE USER WHO HAS IT
C
      CALL LOOK3(IREC(1,30),SEE,FILE,&130,&150)
130   CALL DAY (IREC(1,34),IOUT)
      WRITE(6,131) IX,LIBID,NAME,IOUT
131   FORMAT(//6X,'COPY NO. ',I2,5X,'USER:  ',
     2I5,2X,5A4,5X,'DATE BORROWED',
     3':  ',A4,I2,', 19',I2)
      GO TO 134
132   WRITE(6,133) IX
133   FORMAT(//6X,'COPY NO. ',I2,5X,'IN LIBRARY')
C
C
C CHECK FOR RESERVATION ON THIS COPY OF THE BOOK
C
134   LL = 0
      DO 140 L=31,33
      IF(IREC(1,L).EQ.0)  GO TO 140
      CALL LOOK3(IREC(1,L),SEE,FILE,&135,&140)
135   LL = L - 30
      WRITE(6,136)LL,LIBID,NAME
136   FORMAT(/20X,'RESERVATION NO.',I3,' FOR USER:  ',I5,2X,5A4)
140   CONTINUE
```

```
C
      IF(LL.EQ.0) GO TO 150
C
C  DELETE RESERVATIONS IF REQUIRED
      WRITE(6,142)
142   FORMAT(/' C) DELETE A RESERVATION?')
      READ(5,1000) DEL
C UPDATE THE SYSTEM LOG
      WRITE(7,1000) DEL
1000  FORMAT(A4)
      IF(DEL.NE.KYES) GO TO 150
1002  WRITE(6,1001)
1001  FORMAT(' C) TYPE 1,2, OR 3 TO SPECIFY IT.')
141   READ(5,143,END=150) DEL
C UPDATE THE SYSTEM LOG
      WRITE(7,143) DEL
143   FORMAT(I1)
      IF(DEL.LT.1.OR.DEL.GT.3) GO TO 1002
C
C DELETE RESERVATION AND SHIFT ENTRIES
      DO 144 L=DEL,3
144   IREC(1,L+30) = IREC(1,L+31)
      IREC(1,33) =0
      WRITE(6,1003)
1003  FORMAT(' C) DELETE ANOTHER?')
      READ(5,1000)  DEL
C UPDATE THE SYSTEM LOG
      WRITE(7,1000) DEL
      IF(DEL.EQ.KYES) GO TO 1002
C
C WRITE OUT THE UPDATED LOAN RECORD
145   INDX01 = INDX01 - 1
      FIND(1'1000*INDX01)
      WRITE(1,26) (IREC(1,I),I=1,37)
      INDX01 = INDX01 + 1
C
C
C LAST COPY OF BOOK?
C
150   IF(IX.GE.IREC(1,36))  RETURN 1
C
C NO, THEN GO READ NEXT COPY'S RECORD FROM LOAN FILE
C
      IX = IX + 1
      GO TO 105
C
C IF FIRST TRY AT MATCHING CATALOG NUMBERS, THEN DO NOT COMPARE AUTHORS
155   IF(XFIRST)  GO TO 156
C
C COMPARE AUTHOR NAMES
C
      DO 115 I = 1,24
      IF(IREC(1,I).NE.AUTHOR(I))  GO TO 160
115   CONTINUE
```

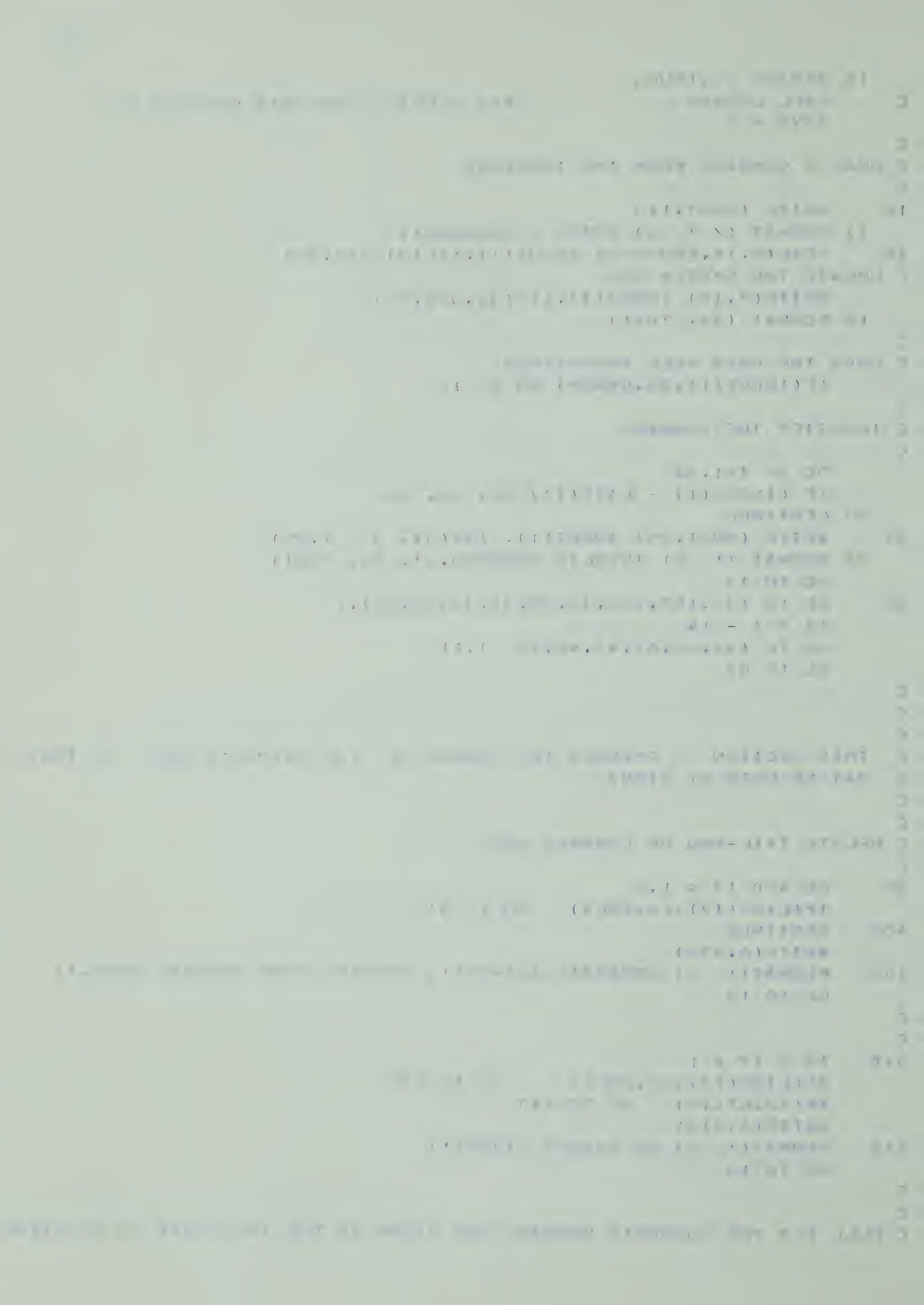


```
      GO TO 157
156   XFIRST = .FALSE.
C
C SET FILE POINTER TO SKIP OVER EXTRA COPIES OF BOOK JUST REJECTED
157   INDX01 = INDX01 + IREC(1,36) - 1
      GO TO 105
C
160   WRITE(6,161)
161   FORMAT(' C) DO YOU HAVE THE RIGHT AUTHOR AND/OR CATALOG NUMBER?')
      RETURN 1
      END



C THIS PROGRAM IS THE MAINLINE FOR THE ON-LINE MODULE 'LIBRARY'.
C              LIBRARY AUTOMATION VERSION 3.2....JUNE 1, 1970
C MTS VERSION OF LIBAUT 3.1
C MTS VERSION..........LAST UPDATED ON OCT. 20, 1970 BY J. DIMSDALE
C
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      DATA PERIOD/'.    '/,COMMA/',    '/
      DATA QMARK/'?    '/
      DATA SCORE/'_    '/
      DATA STAR/'*    '/
      DATA INPT/5/
      DIMENSION IOUT(3),NUMBER(15),LINE(76),AUTHOR(24)
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40),INDX26
     X /COMAND/ LIST(22),KBLA,KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      COMMON/PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL,USTOR1,QSTOR1,FSIN24,
     CALFLAG,MAXPTR,INDX01,INDX02,
     CINDX03,INDX04,INDX08,INDX20,INDX21,INDX22,INDX23,INDX24,INDX25,
     CMAXAUT
      INTEGER SUPRES /Z24000000/, RESTOR /Z14000000/
C
      EQUIVALENCE (NUMBER(1),IF(800)),
     C            (LINE(1),IF(1)),
     C            (AUTHOR(1),INPUT(3)),
     C            (NUMSUR,INPUT(32))
C
C
C
C                    IDENTIFICATION OF COMMAND WORDS
C
C   LIST(1)   SEARCH                LIST(9)    FIND
C        2    BRING                      10    DEMAND
C        3    RESERVE                    11    SAVE
C        4    ENQUIRE                    12    COPY
```

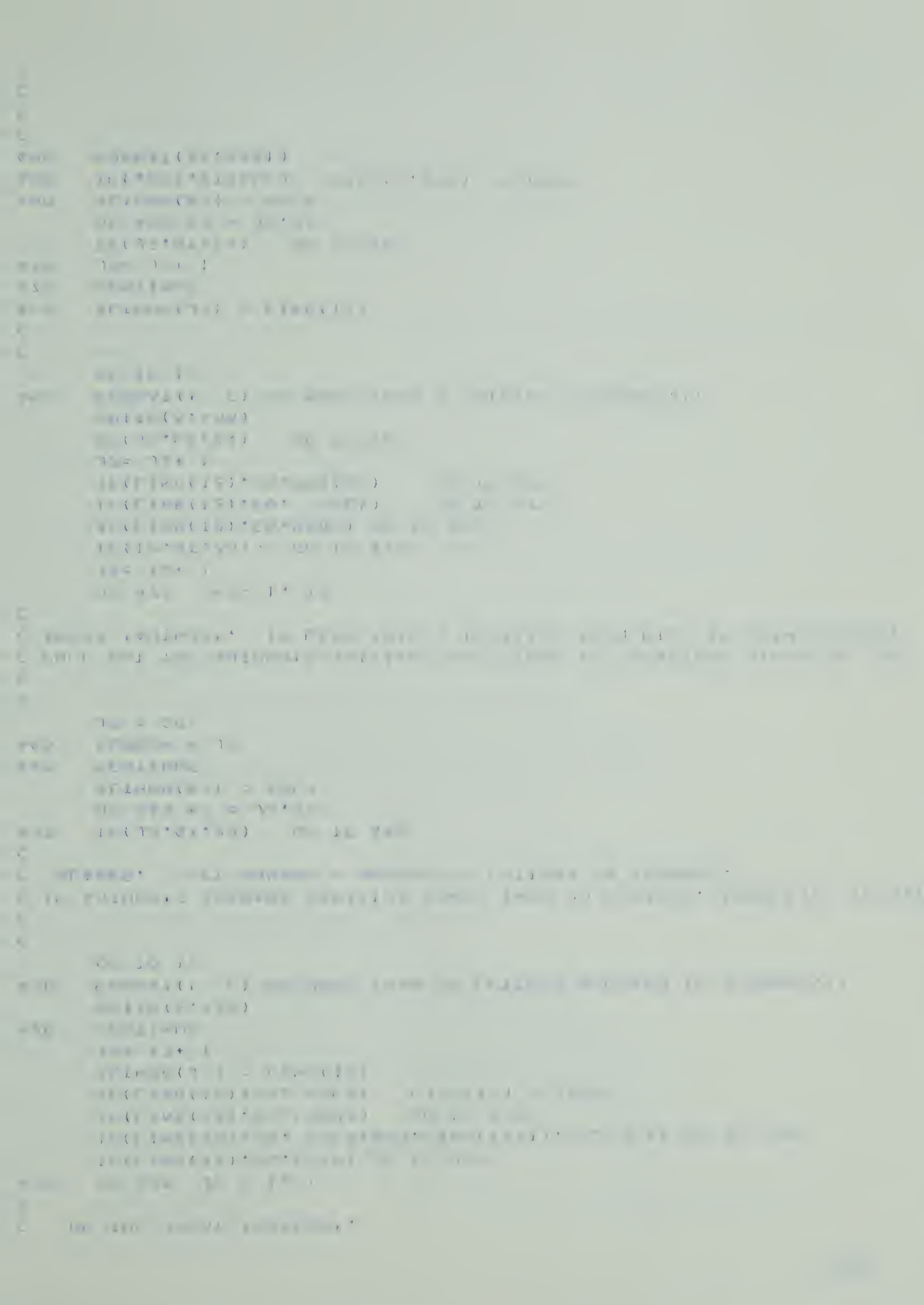


```
C         5     LOAN                          13     DISPLAY
C         6     RETURN                        14     DAY
C         7     REQUEST                       15     LIST
C         8     RENEW                         16     STOP
C
C        17     ENROLL                        18     LOOKUP
C        19     ALTER                         20     REMOVE
C        21     AVAILABLE                     22     TRACE
C
C  INPUT(1 TO 2)          = COMMAND WORD (2A4)
C  INPUT(3 TO 26)         = AUTHOR NAME (24A1)
C  INPUT(27 TO 30)        = CATALOG NUMBER 4A4
C  INPUT(31)              = USER'S ID (I6)
C
C
C
      WRITE (LIB, 300)
  300 FORMAT ('1   LIBRARY AUTOMATION...VERSION 3.2...JUNE 1, 1970.')
C
C OBTAIN LENGTH OF FILE 'AUTHORS'
C
      CATLOG = 3
      READ(CATLOG'1000,3) MAXAUT
3     FORMAT(I5)
C
C SETUP THE IN-CORE TABLES
      CALL SETUP(&5)
C
C OBTAIN TODAY'S DATE
C DATE IS PLACED IN DSRN 3 ('CATALOG') BY MTS COMMAND 'TIME'
C
5     READ(3'3000,4) IOUT
4     FORMAT(23X,3(1X,I2))
      CALL DATE(NDATE,IOUT)
      GO TO 6
C
C
    1 WRITE (LIB,2)
    2 FORMAT (// '  C) ENTER TODAY''S NUMBER')
      READ (5,55,END=1) (LAST(J), J = 1, 4)
C UPDATE THE SYSTEM LOG
      WRITE(7,55) (LAST(J),J=1,4)
55    FORMAT(80A1)
      CALL CHECK (LIB, LAST, 1, 4, NDATE, .FALSE., &1)
      CALL DAY (NDATE, IOUT)
    6 WRITE (LIB,8) IOUT
    8 FORMAT ('  C) ', A4, I2, ', 19', I2 )
      D = NDATE
      IF(NDATE.GE.1000 ) D = NDATE/10
      ID = (D**2-3*D+1)/10
      WRITE(6,1500)       ID
1500  FORMAT(1X,I10)
C
C
```

```
   10 XTRACE = .TRUE.
C      CALL LOGDSK                ***** DELETED FROM MTS VERSION ***
       TRYS = 0
C
C READ A COMMAND FROM THE TERMINAL
C
12     WRITE (NOUT,11)
   11 FORMAT (/ '  C) ENTER A COMMAND')
15     READ(5,16,END=200) INPUT(1),(IF(J),J=1,76)
C UPDATE THE SYSTEM LOG
       WRITE(7,16) INPUT(1),(IF(J),J=1,76)
   16 FORMAT (A4, 76A1)
C
C DOES THE USER NEED PROMPTING?
       IF(INPUT(1).EQ.QMARK) GO TO 12
C
C IDENTIFY THE COMMAND
C
       DO 20 I=1,22
       IF (INPUT(1) - LIST(I)) 20, 30, 20
   20 CONTINUE
24     WRITE (NOUT,25) INPUT(1), (IF(J), J = 1,76)
   25 FORMAT ('  C) INVALID COMMAND..', A4, 76A1)
       GO TO 15
30     GO TO (35,155,160,35,35,35,35,35,35),I
       II = I - 14
       GO TO (48,62,62,48,48,48  ),II
       GO TO 62
C
C
C
C  THIS SECTION    CHANGES THE FORMAT OF  THE AUTHOR'S NAME  SO THAT IT
C  MAY BE USED BY BINS3
C
C
C DELETE TAIL-END OF COMMAND WORD
C
35     DO 400 I2 = 1,6
       IF(LINE(I2).EQ.KBLA)   GO TO 410
400    CONTINUE
       WRITE(6,405)
405    FORMAT('  C) SEPARATE AUTHOR''S SURNAME FROM COMMAND WORD.')
       GO TO 10
C
C
410    I2 = I2 + 1
       IF(LINE(I2).NE.KBLA)      GO TO 420
       IF(I2.LT.16)   GO TO 410
       WRITE(6,415)
415    FORMAT('  C) NO AUTHOR GIVEN.')
       GO TO 10
C
C
C PULL OUT THE AUTHOR'S SURNAME AND STORE IN THE THE FIRST 20 LOCATIONS
```

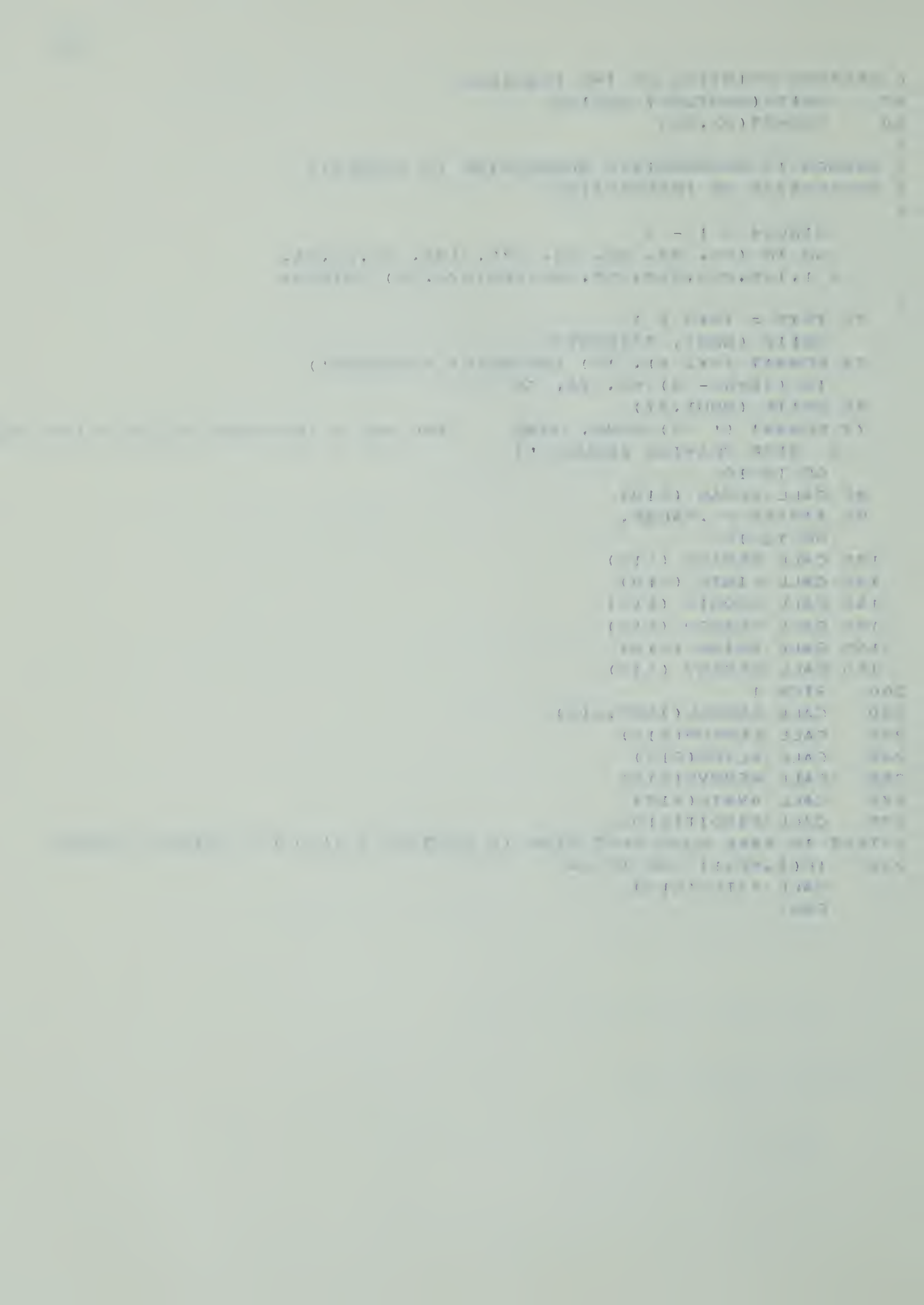


```
C    OF THE ARRAY 'AUTHOR'.
C
420    DO 425  J2 = 1,21
       IF(LINE(I2).EQ.STAR) GO TO 280
       IF(LINE(I2).EQ. KBLA.AND.LINE(I2+1).EQ.KBLA) GO TO 435
       IF(LINE(I2).EQ.COMMA)    GO TO 435
       IF(LINE(I2).EQ. KBLA)    LINE(I2) = SCORE
       AUTHOR(J2) = LINE(I2)
       I2= I2+ 1
425    CONTINUE
       WRITE(6,430)
430    FORMAT('  C) NO MORE THAN 20 LETTERS ALLOWED IN SURNAME.')
       GO TO 10
C
C
C IF AUTHOR'S SURNAME CONTAINS FEWER THAN 20 LETTERS, THEN FILL IN WITH
C  BLANKS.    SET NUMSUR = NUMBER OF LETTERS IN SURNAME.
C
435    IF(J2.GT.20)    GO TO 445
       DO 440 K2 = J2,20
       AUTHOR(K2) = KBLA
440    CONTINUE
445    NUMSUR = J2
       J2 = 20
C
C
C PULL OUT THE AUTHOR'S INITIALS AND STORE IN LOCATIONS 21-24 OF  THE
C ARRAY 'AUTHOR'.  IF LESS THAN 4 INITIALS THEN FILL IN WITH BLANKS.
C
       DO 470    K2= 1, 15
       I2= I2+ 1
       IF(I2.GT.76)    GO TO 475
       IF(LINE(I2).EQ.STAR) GO TO 280
       IF(LINE(I2).EQ.  KBLA)     GO TO 470
       IF(LINE(I2).EQ.PERIOD)     GO TO 470
       J2= J2+ 1
       IF(J2.LE.24)    GO TO 460
       WRITE(6,455)
455    FORMAT('  C) NO MORE THAN 4 INITIALS ALLOWED.')
       GO TO 10
C
C
460    AUTHOR(J2) = LINE(I2)
470    CONTINUE
475    J2= J2+ 1
       IF(J2.GT.24)    GO TO 485
       DO 480 K2 = J2,24
480    AUTHOR(K2) = KBLA
485    IF(.NOT.XTRACE)  WRITE(6,490)  AUTHOR
490    FORMAT(1X,24A1)
C
C
C
C
```

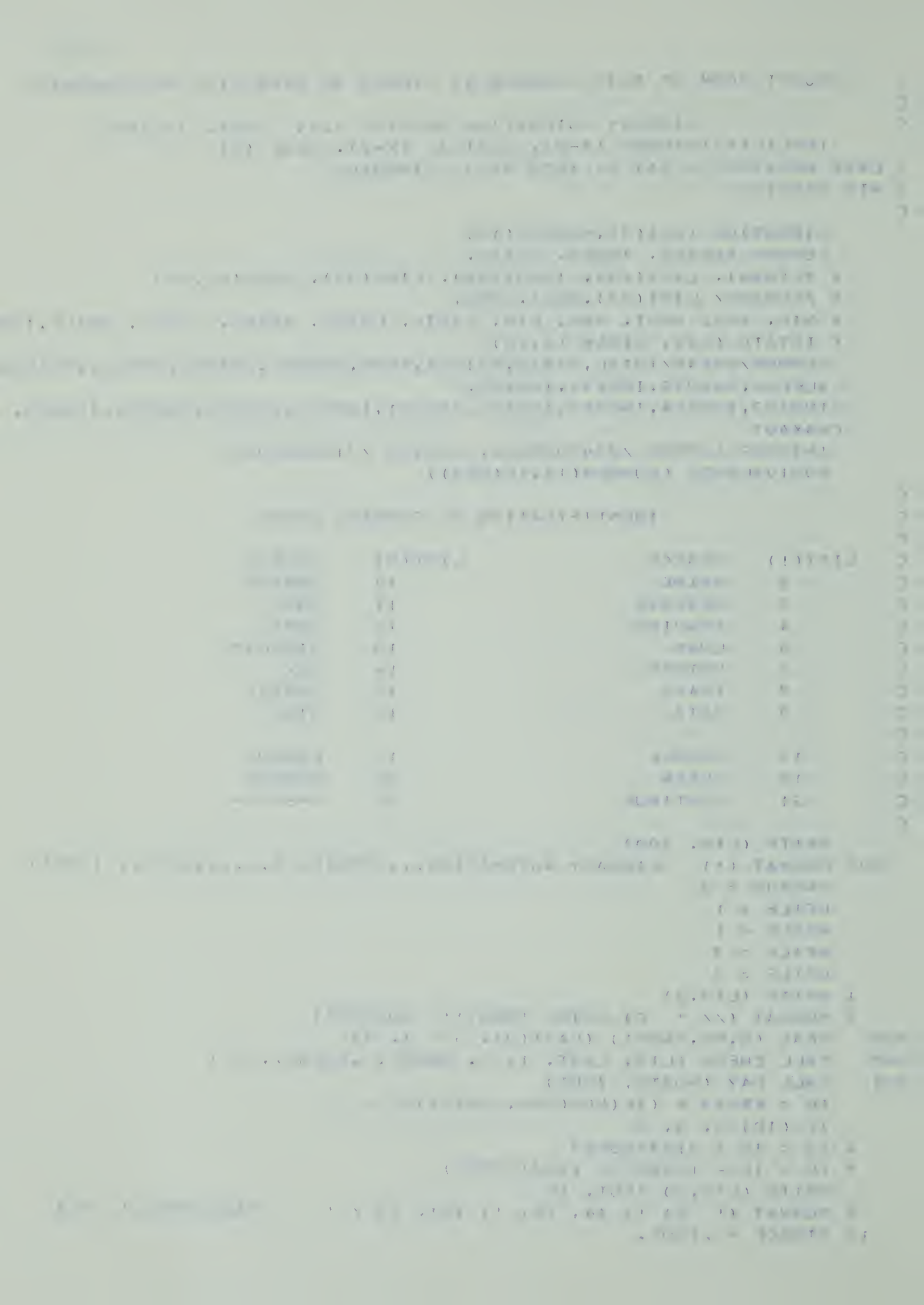


```
C SEARCH THE AUTHOR INDEX FILE 'AUTHORS' AND SET POINTERS
C FOR FILES 'LOANS' AND 'CATALOG'
C
      XSEAR = I.EQ.1
      CALL QUICK(AUTHOR,XSEAR,&10)
      IF(XSEAR)    GO TO 150
      GO TO 44
C
C OBTAIN CATALOG NUMBER
C
   40 WRITE (NOUT, 41)
   41 FORMAT ('  C) ENTER CATALOG NUMBER')
44    READ (NIN,45,END=40) (INPUT(J), J = 27, 30)
C UPDATE THE SYSTEM LOG
      WRITE(7,45) (INPUT(J),J=27,30)
   45 FORMAT (4A4)
C
C DOES THE USER REQUIRE PROMPTING?
      IF(INPUT(27).EQ.QMARK) GO TO 40
C
      CALL CHECK (NOUT, INPUT, 27, 30, DUMMY, .TRUE., &40)
      IF(I.EQ.9) GO TO 62
C
C OBTAIN USER ID
C
48    SEE = 0
50    CALL  IDENT(INPUT(31),SEE,&10)
C
      II = I - 3
      GO TO (145,62,62, 80,62), II
      GO TO 62
C
C OBTAIN PASSWORD
C
60    WRITE(6,61) RESTOR
61    FORMAT(1X,A1,'  C) ENTER PASSWORD.')
C SUPPRESS PRINTING ON THE TERMINAL
62    WRITE(6,68)  SUPRES
65    READ (LIBIN,55,END=60) (LAST(J), J = 1, 5)
C UPDATE THE SYSTEM LOG
      WRITE(7,55) (LAST(J),J=1,5)
C
C DOES THE USER REQUIRE PROMPTING?
      IF(LAST(1).EQ.QMARK) GO TO 60
C
C CONVERT CHARACTER FORM OF PASSWORD NUMBER TO
C INTERNAL BINARY FORM
C
      CALL CHECK (LIB, LAST, 1, 5, PASS, .FALSE., &70)
      INDEX = 0
C
C IS THE PASSWORD CORRECT?
C
      IF (PASS .NE. ID) GO TO 70
```

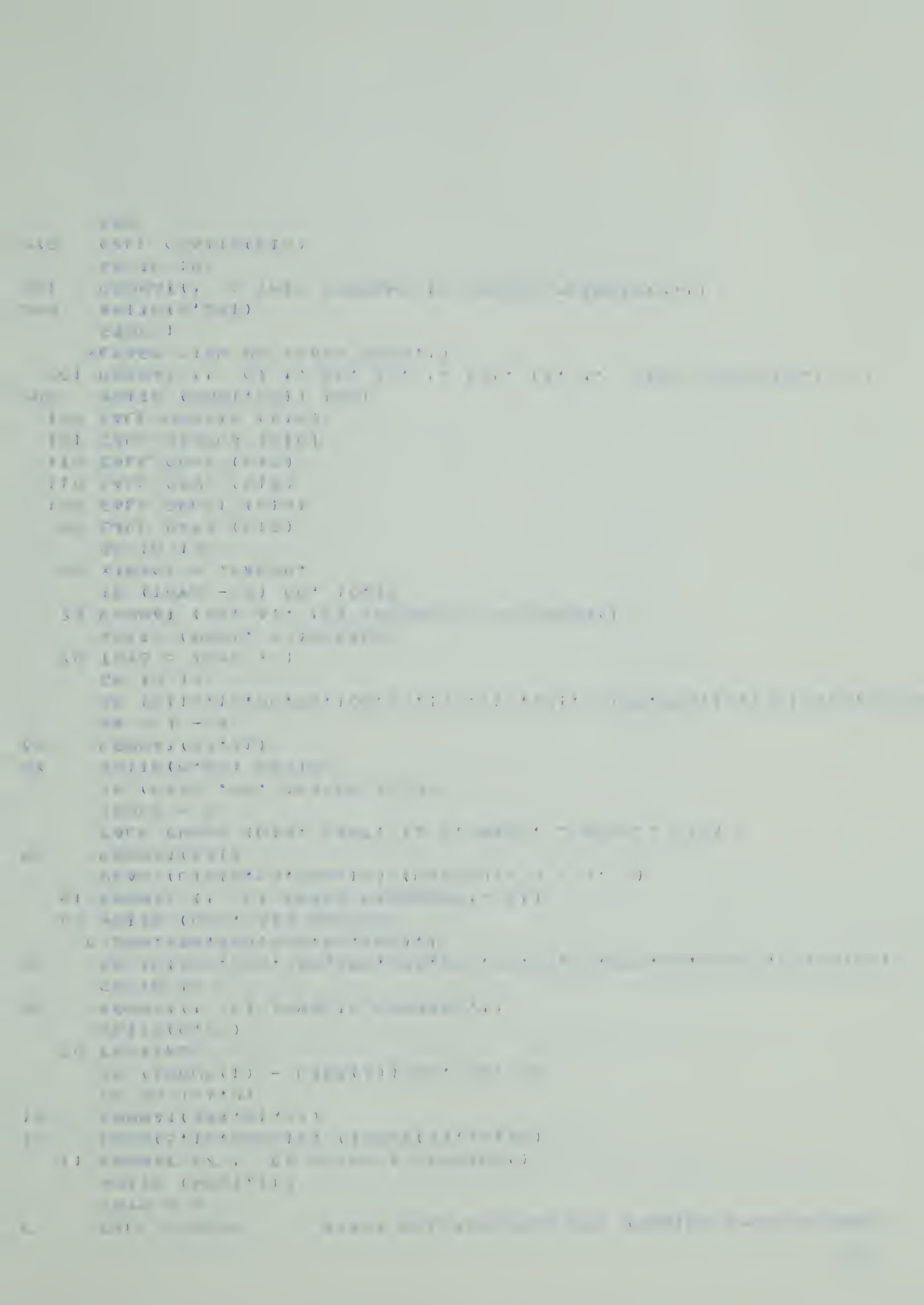
```
C RESTORE PRINTING ON THE TERMINAL
67     WRITE(NOUT,68) RESTOR
68     FORMAT(2X,A1)
C
C BRANCH TO APPROPRIATE SUBROUTINE TO COMPLETE
C PROCESSING OF TRANSACTION
C
       MINUS4 = I - 4
       GO TO (80, 80, 80, 80, 275, 105, 24,24,24,
      X 1,125,200,250,255,260,265,270,90), MINUS4
C
    70 TRYS = TRYS + 1
       WRITE (NOUT, 73)RESTOR
    73 FORMAT (2X, A1, 'C) INCORRECT PASSWORD')
       IF (TRYS - 3) 60, 76, 76
    76 WRITE (NOUT,77)
    77 FORMAT ('  C) DUMB, DUMB      YOU HAD 3 TRYS AND YOU BLEW THEM ALL
      X  STOP PLAYING AROUND ')
       GO TO 10
    80 CALL ALOAN (&10)
    90 XTRACE = .FALSE.
       GO TO 15
   105 CALL DEMAND (&10)
   125 CALL LIST2 (&10)
   145 CALL ENQUIR (&10)
   150 CALL SEARCH (&10)
   155 CALL BRING (&10)
   160 CALL RESERV (&10)
200    STOP 1
250    CALL ENROLL(INPT,&10)
255    CALL LOOKUP(&10)
260    CALL ALTER(&10)
265    CALL REMOVE(&10)
270    CALL AVAIL(&10)
275    CALL FINDIT(&10)
C TEST TO MAKE SURE THAT STAR IS USED ONLY WITH THE SEARCH COMMAND
280    IF(I.NE.1)  GO TO 24
       CALL TITLE(&10)
       END
```



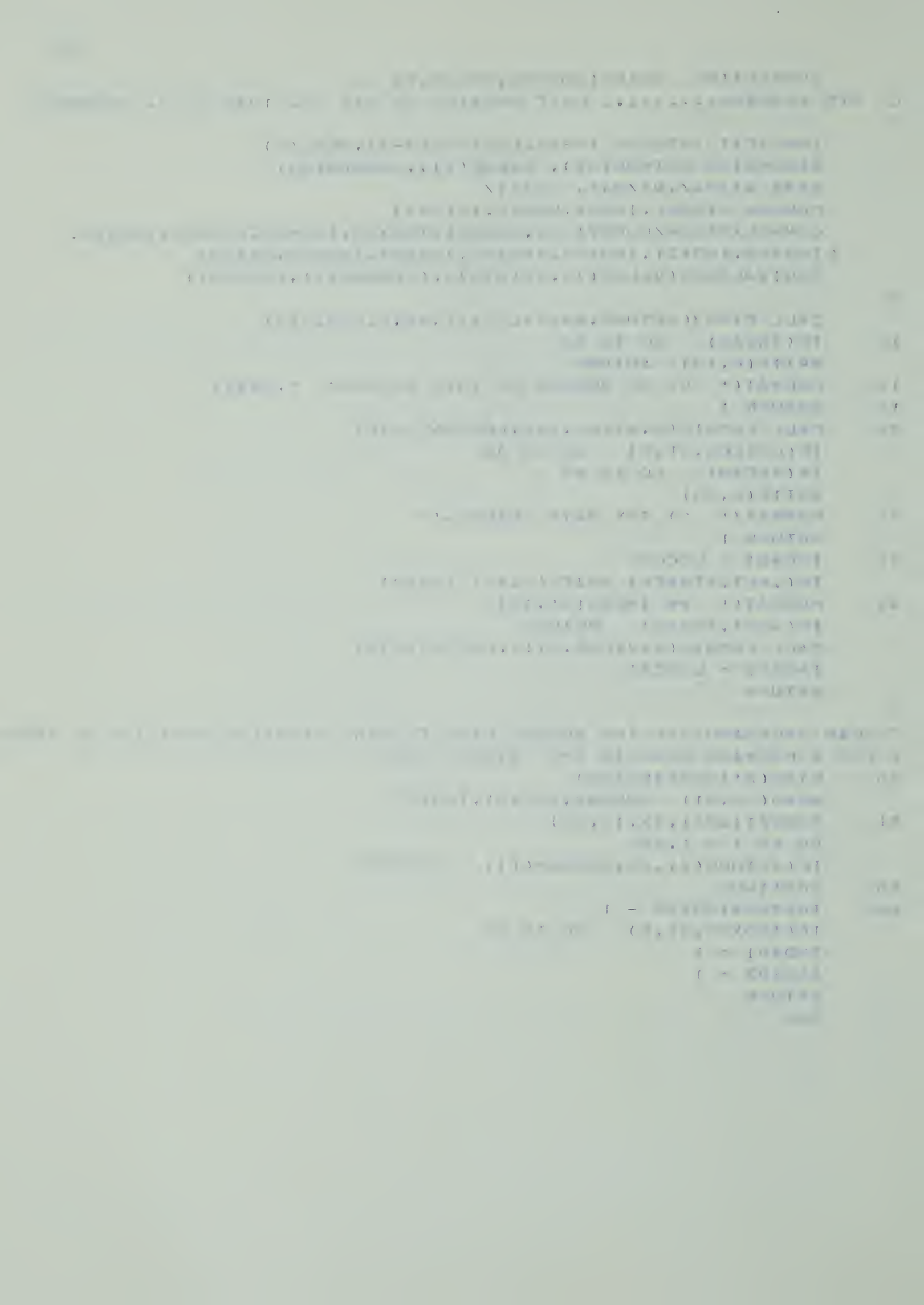
```
C     OBJECT FORM OF THIS PROGRAM IS STORED IN DISK FILE "MAINUPDATE"
C
C                  LIBRARY AUTOMATION VERSION 3.1,  SEPT. 30,1969
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
C LAST UPDATED ON MAY 6, 1970 BY J. DIMSDALE
C MTS VERSION
C
      DIMENSION IOUT(3),NUMBER(15)
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22),KBLA,KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      COMMON/PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL,USTOR1,OSTOR1,FSIN24,
     CALFLAG,MAXPTR,INDX01,INDX02,
     CINDX03,INDX04,INDX08,INDX20,INDX21,INDX22,INDX23,INDX24,INDX25,
     CMAXAUT
      INTEGER SUPRES /Z24000000/, RESTOR /Z14000000/
      EQUIVALENCE (NUMBER(1),IF(800))
C
C                    IDENTIFICATION OF COMMAND WORDS
C
C   LIST(1)    SEARCH                LIST(9)     DELETE
C        2     BRING                       10     DEMAND
C        3     RESERVE                     11     SAVE
C        4     ENQUIRE                     12     COPY
C        5     LOAN                        13     DISPLAY
C        6     RETURN                      14     DAY
C        7     TRACE                       15     UPDATE
C        8     DATA                        16     STOP
C
C       17     ENROLL                      18     LOOKUP
C       19     ALTER                       20     REMOVE
C       21     CONTINUE                    22     -------
C
      WRITE (LIB, 300)
  300 FORMAT ('1   LIBRARY AUTOMATION...VERSION 3.2...JUNE 1, 1970')
      INDX08 = 2
      LFILE = 1
      MFILE = 1
      NFILE = 3
      OFILE = 3
    1 WRITE (LIB,2)
    2 FORMAT (// '  C) ENTER TODAY''S NUMBER')
  888 READ (5,55,END=1) (LAST(J), J = 1, 4)
  890 CALL CHECK (LIB, LAST, 1, 4, NDATE, .FALSE., &1)
  891 CALL DAY (NDATE, IOUT)
      ID = 65539 * (IF(MOD(500,NDATE))*2 + 1)
      IF (ID) 4, 4, 5
    4 ID = ID + 2147483647
    5 ID = ID - 100000 * (ID/100000)
      WRITE (LIB,8) IOUT, ID
    8 FORMAT ('  C) ', A4, I2, ', 19', I2 / '      PASSWORD..', I5)
   10 XTRACE = .TRUE.
```



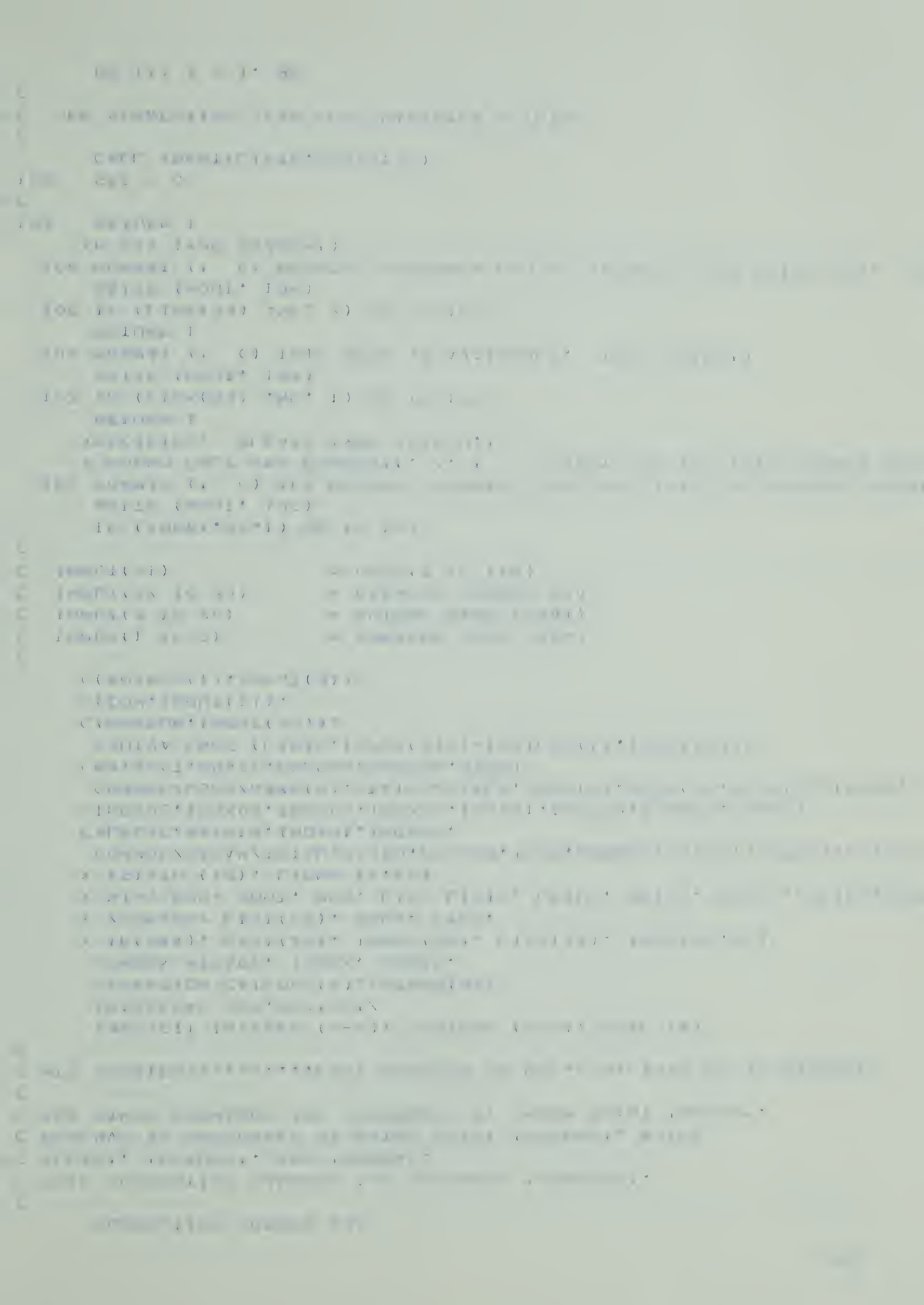
```
C       CALL LOGDSK          ***** DELETED FROM MTS VERSION ************
        TRYS = 0
        WRITE (NOUT,11)
   11 FORMAT (/ '  C) ENTER A COMMAND')
15      READ(5,16,END=10) (INPUT(I),I=1,5)
16      FORMAT(3A4,A1,A4)
        DO 20 I=1,21
        IF (INPUT(1) - LIST(I)) 20, 30, 20
   20 CONTINUE
        WRITE(6,25)
25      FORMAT('  C) INVALID COMMAND.')
        GO TO 15
30      GO TO(250,250,250,250,250,250,60,60,60,250,60,60,60,60,60,60,
      C 250,250,250,250,60,250),I
   60 WRITE (LIB, 61) SUPRES
   61 FORMAT ('  C) ENTER PASSWORD', A1)
        READ (LIBIN,55,END=10) (LAST(J), J = 1, 5)
55      FORMAT(6A1)
        CALL CHECK (LIB, LAST, 1, 5, PASS, .FALSE., &70)
        INDEX = 0
        IF (PASS .NE. ID) GO TO 70
67      WRITE(6,68) RESTOR
68      FORMAT(2X,A1)
        M4 = I - 4
        GO TO(10,10,90,95,100,10,110,115,121,1,125,200,10,10,10,10,270),M4
        GO TO 10
   70 TRYS = TRYS + 1
        WRITE (NOUT, 73)RESTOR
   73 FORMAT (2X, A1, 'C) INCORRECT PASSWORD')
        IF (TRYS - 3) 60, 10,10
   90 XTRACE = .FALSE.
        GO TO 15
   95 CALL DATA (&10)
  100 CALL DELET (&10)
  110 CALL SAVE (&10)
  115 CALL COPY (&10)
  121 CALL DISPLA (&10)
  125 CALL UPDATE (&10)
200     WRITE (NOUT,201) IOUT
  201 FORMAT ('  C) ', A4, I2, ', 19', I2, '.  STOP EXECUTED.' / '      P
      XLEASE SIGN ON AGAIN SOON.')
        STOP 1
250     WRITE(6,251)
251     FORMAT('  C THIS COMMAND IN MODULE "LIBRARY".')
        GO TO 10
270     CALL CONTIN(&10)
        END
```



```
      SUBROUTINE  QUICK(AUTHOR,XSEAR,*)
C  MTS VERSION......... LAST UPDATED ON MAY 20, 1970 BY J. DIMSDALE
C
      IMPLICIT INTEGER (A-W),LOGICAL(X-Z),REAL($)
      DIMENSION AUTHOR(24), VALUE(11),SURNAM(20)
      DATA WA/24/,V1/24/, WV/11/
      COMMON XTRACE,INDEX,NDATE,IF(844)
      COMMON/PARAM/DUMMY(10),INDX01,INDX02,INDX03,INDX04,INDX08,
     CINDX20,INDX21,INDX22,INDX23,INDX24,INDX25,MAXAUT
      EQUIVALENCE(VALUE(1),IF(830)),(SURNAM(1),IF(800))
C
      CALL BINS3(AUTHOR,WA,VALUE,V1,WV,K,&20,&10)
10    IF(XSEAR)   GO TO 60
      WRITE(6,15)  AUTHOR
15    FORMAT('  C) NO RECORD OF THIS AUTHOR:  ',24A1)
16    RETURN 1
20    CALL EBCDIC(6,VALUE,1,5,LOCLON ,&16)
      IF(LOCLON.GT.0)   GO TO 40
      IF(XSEAR)   GO TO 40
      WRITE(6,30)
30    FORMAT('  C) TRY MAIN AUTHOR.')
      RETURN 1
40    INDX01 = LOCLON
      IF(.NOT.XTRACE) WRITE(6,43) INDX01
43    FORMAT('  ** INDX01=',I6)
      IF(.NOT.XSEAR)   RETURN
      CALL EBCDIC(6,VALUE,6,11,LOCCAT,&16)
      INDX03 = LOCCAT
      RETURN
C
C SCAN BACKWARDS IN THE AUTHOR FILE TC FIND STARTING POSITION OF INDX03
C FOR A FORWARD SCAN IN THE CATALOG FILE
50    FIND(2'1000*INDX25)
      READ( 2,51)  SURNAM,INDX01,INDX03
51    FORMAT(20A1,4X,I5,I6)
      DO 55 I = 1,20
      IF(AUTHOR(I).NE.SURNAM(I))   RETURN
55    CONTINUE
60    INDX25=INDX25 - 1
      IF(INDX25.GT.0)   GO TO 50
      INDX01 = 1
      INDX03 = 1
      RETURN
      END
```



```
      SUBROUTINE RESERV (*)
C
C THIS SUBROUTINE HANDLES THE COMMANDS 'RESERVE',
C 'LOAN', 'RETURN', AND 'RENEW'.
C RESERVE IS PERFORMED BY ENTRY POINT 'RESERV', WHILE
C THE OTHER COMMANDS ARE PERFORMED BY ENTRY POINT 'ALOAN'.
C
C MTS VERSION.........LAST UPDATED ON OCT. 20, 1970 BY J. DIMSDALE
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      INTEGER*2 COM,RE/'RE'/
      DIMENSION CATLOG (4),AUTHOR(24)
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), LAST(20), INPUT(36), LINK(34), IREC(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      COMMON/PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL,USTOR1,OSTOR1,FSIN24,
     CALFLAG,MAXPTR,INDX01,INDX02,
     CINDX03,INDX04,INDX08,INDX20,INDX21,INDX22,INDX23,INDX24
      COMMON/USER/NAME(5),STATUS,TITLE,ADDR(6),PHON(2),DEPT(2),IDNUM,K1,
     CK2,ACCT,EDATE,ONLOAN,OVRDUE,OVER1
      EQUIVALENCE (LIBID,INPUT(31)),(CATLOG(1),INPUT(27)),
     C(NUMSUR,INPUT(32)),
     C(COM,INPUT(1)),
     C(AUTHOR(1),INPUT(3))
C
C  INPUT(1 TO 2)          = COMMAND WORD (2A4)
C  INPUT(3 TO 26)         = AUTHOR NAME (24A1)
C  INPUT(27 TO 30)        = CATALOG NUMBER 4A4
C  INPUT(31)              = USER'S ID (I6)
C
      IF (INDEX.EQ.1) GO TO 103
      WRITE (NOUT, 102)
  102 FORMAT ('  C) THE RESERVE COMMAND REQUIRES THAT THE SEARCH COMMAND
     X FOUND ONLY ONE CORRECT', /, '     ENTRY FOR THE TITLE WORDS YOU S
     XPECIFIED.  PLEASE TYPE SEARCH.')
      RETURN 1
  103 IF (LINK(33) .NE. 1) GO TO 105
      WRITE (NOUT, 104)
  104 FORMAT ('  C) THIS BOOK IS AVAILABLE.  TYPE BRING')
      RETURN 1
  105 IF (LINK(33) .NE. 7) GO TO 108
      WRITE (NOUT, 106)
  106 FORMAT ('  C) NO MORE RESERVES CAN BE ACCEPTED FOR THIS BOOK.  EAC
     XH DAY TYPE SEARCH')
107   RETURN 1
C
108   SEE = 0
      CALL IDENT(LIBID,SEE,&107)
C
C  SEE SUBROUTINE SEAR FOR CHARACTER OF LINK
C
      DO 111 I = 1, 28
```



```
      INPUT(I+2) = LINK(I)
  111 CONTINUE
C
      XSEAR = .FALSE.
      CALL QUICK(AUTHOR,XSEAR,&107)
C
C
C  THE FOLLOWING PART OF THIS PROGRAM ALSO HANDLES LOANS AND RETURNS
C
C
      ENTRY ALOAN (*)
C
C  STRUCTURE OF MFILE
C    (24A1)    AUTHOR
C    (I2)      LENGTH OF AUTHOR SURNAME
C    (4A4)     CATALOG NUMBER
C    (4I6)     ID NUMBERS OF 4 USERS
C    (I4)      DAY FIRST USER BORROWED BOOK.  (IF BOOK NOT OUT, DAY NUMBE
C              BOOK FIRST BECAME AVAILABLE TO REQUESTOR)
C    (I3)      NUMBER OF TIMES THIS COPY HAS BEEN BORROWED INCLUDING PRES
C    (I2)      NUMBER OF COPIES
C    (I2)      NUMBER OF DAYS BOOK MAY BE BORROWED
C
C
      IF(XTRACE) GO TO 16
14    WRITE(6,15) MFILE,INDX01
15    FORMAT(' **STARTING TRACE IN RESERV...'/' **MFILE=',I2,' INDX01=',
     XI5)
16    MFILE = 1
      IF(.NOT.XTRACE) WRITE(6,18) MFILE
18    FORMAT(1X,'**MFILE=',I4)
      XTRANS = .TRUE.
      XRENEW = INPUT(1).EQ.LIST(8)
      XRET = INPUT(1).EQ.LIST(6).OR.XRENEW
      XREQ = COM.EQ.RE
      INDEX = 0
C
      FIND (MFILE'1000*INDX01)
   20 IX = 1
   25 READ (MFILE, 26,END =80) (IREC(IX, I), I = 1, 37)
   26 FORMAT (24A1, I2, 4A4, 4I6, I4, I3, 2I2)
      INDX01 = INDX01 + 1
      IF(.NOT.XTRACE)  WRITE(6,27) (IREC(IX,I),I=1,37 )
   27 FORMAT(1X,24A1,I2,4A4,4I6,I4,I3,2I2)
C
C  COMPARE AUTHOR SURNAMES
C
993   DO 30 ID = 1, NUMSUR
      IF(IREC(IX,ID) -AUTHOR(ID)) 66,30,79
   30 CONTINUE
C
C  COMPARE CATALOG NUMBERS
C
      DO 35 IE = 1, 4
```

```
      IF(IREC(IX,IE+25).NE.CATLOG(IE))    GO TO 66
   35 CONTINUE
C
C CHECK FOR REFERENCE STATUS OF BOOK
C
      IF(IREC(IX,37).NE.0) GO TO 37
      WRITE(6,36)
36    FORMAT('  C) REFERENCE BOOK...CANNOT LEND')
      RETURN 1
C
C CHECK FOR RETURN OR RENEWAL
C
37    IF(XRET) GO TO 95
      IF (IREC(IX,30).EQ.0) GO TO 85
C
C
C CHECK TO SEE IF USER HAS BOOK ON LOAN ALREADY
C
      IF(IREC(IX,30).EQ.LIBID) GO TO 500
C
   39 IF (IX.GE.IREC(1,36)) GO TO 41
      IX = IX + 1
      GO TO 25
C
C
C IF USER HAS SPECIAL STATUS THEN ALLOW DUPLICATE TRANSACTION
500   IF(STATUS.EQ.8) GO TO 39
      WRITE(6,501)
501   FORMAT(' C) SORRY, BUT THIS USER CANNOT BORROW THIS BOOK AS HE/SHE
     1 ALREADY HAS IT ON LOAN.')
      RETURN 1
C
C
C CHECK FOR REQUEST OR RESERVATION
C
41    IF (XREQ)    GO TO 45
      WRITE (NOUT, 42)
   42 FORMAT ('  C) THIS BOOK HAS ALREADY BEEN LOANED.  DO YOU WISH TO R
     XESERVE IT?')
      READ (NIN, 43,END=41) IREPLY
C UPDATE THE SYSTEM LOG
      WRITE(7,43) IREPLY
   43 FORMAT (A4)
      IF (IREPLY.NE.KYES) RETURN 1
C
C CHECK FOR PREVIOUS LOAN OR RESERVATION
C
45    DO 46 I = 30,33
      DO 46 J = 1,IX
      IF(IREC(J,I).EQ.LIBID) GO TO 52
46    CONTINUE
C
C LOCATE EARLIEST VACANT SLOT
C
```

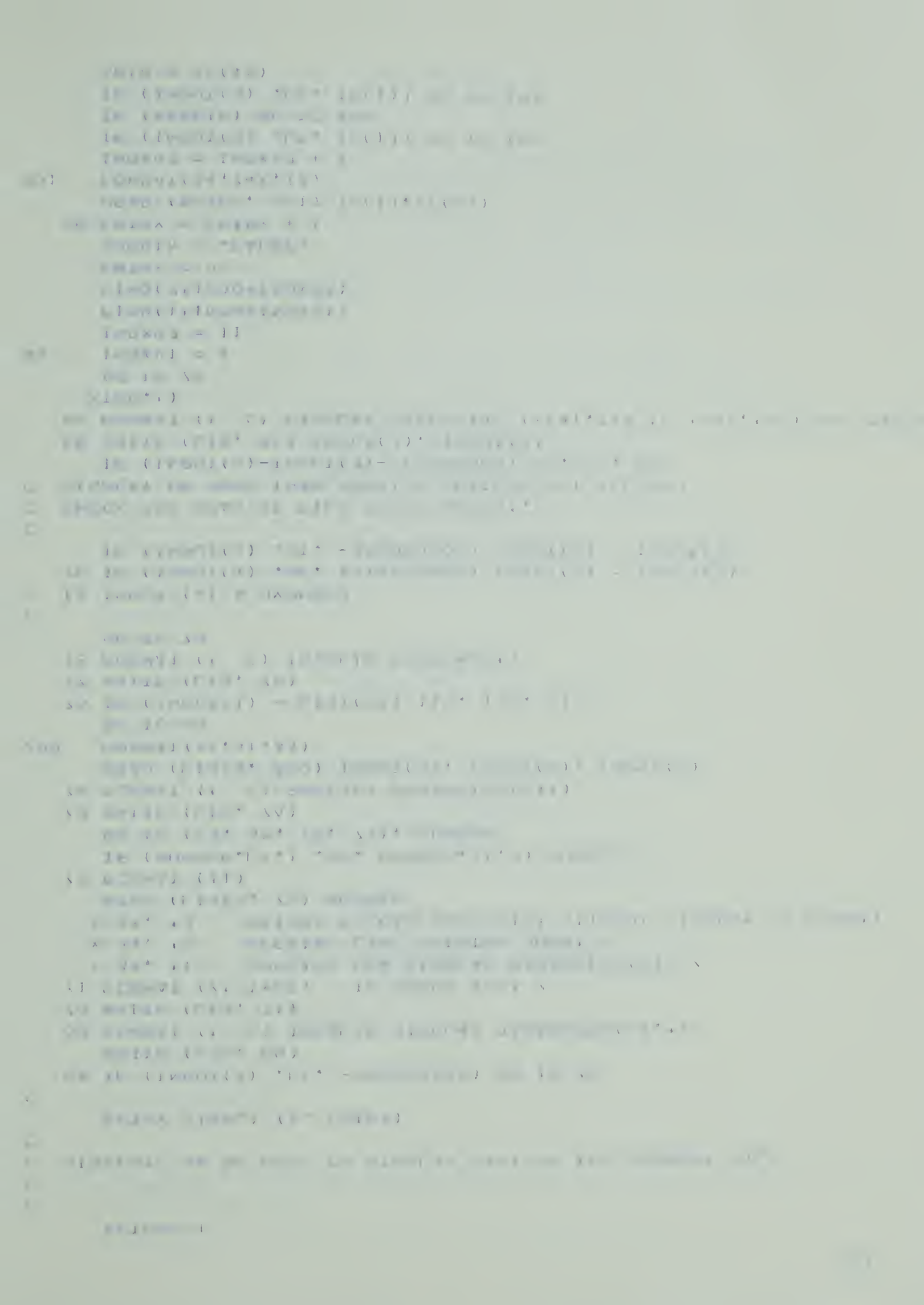
```
49      DO 50 I = 31,33
        DO 50 J = 1, IX
        IF(IREC(J,I).EQ.0) GO TO 55
   50 CONTINUE
        WRITE (NOUT, 106)
        RETURN 1
C
C IF USER HAS SPECIAL STATUS THEN ALLOW DUPLICATE TRANSACTION
52      IF(STATUS.EQ.8) GO TO 49
        WRITE(6,54)
54      FORMAT('  C) THIS RESERVATION CANNOT BE ACCEPTED AS YOU ALREADY HA
       CVE THIS BOOK ON LOAN OR ON RESERVE.' )
        RETURN 1
C
C ENTER RESERVATION
C
55      IREC(J,I)=LIBID
C
C SET LOAN FILE POINTER TO THE CORRECT COPY OF THE BOOK
        INDX01 = INDX01 - IREC(1,36) + J
C
C DETERMINE TIME REMAINING ON THE CURRENT LOAN FOR THIS COPY OF THE BOOK
        LEFT = IREC(J,34) + IREC(1,37) - NDATE
        PLACE=I-30
        WRITE(6,58) PLACE,J,LEFT
58      FORMAT(' C) YOUR RESERVATION IS NO.',I2,' ON COPY NO.',I3,' WITH',
       1I4,' DAYS REMAINING ON ITS CURRENT LOAN.')
        WRITE (NOUT, 57) (INPUT(IP), IP = 3, 31)
   57 FORMAT('  C) EACH DAY TYPE:'/5X,'ENQUIRE ',24A1/5X,4A4/5X,I6)
C
C  IF ID NUMBER ENTERED IN IREC, INDEX = 1
C
   60 INDEX = 1
        IF (.NOT.XTRACE)
      X WRITE (6, 62) ((IREC(IQ,IR), IR = 1, 37), IQ = 1, IX)
   62 FORMAT (' THIS BOOK''S ENTRY:', /, (1X, 24A1, I2, 4A4, 4I6, I4,
      X I3, 2I2))
C
C WRITE OUT LOAN FILE ENTRY
C
65      INDX01 = INDX01 - 1
        FIND (MFILE'1000*INDX01)
        WRITE(MFILE,26) (IREC( J,IR),IR= 1,37)
C
66      IF (INDEX.NE.1) GO TO 20
        WRITE (NOUT, 75)
   75 FORMAT ('  C) TRANSACTION COMPLETED')
        RETURN 1
C
79      IF(IREC(IX,ID).EQ.KBLA)   GO TO 66
C
80      IF(XTRANS)  GO TO 83
        WRITE(6,82)
82      FORMAT('  C) NO RECORD OF THIS LOAN OR RESERVATION.')
```

```
      RETURN 1
C
   83 WRITE (NOUT, 84)
   84 FORMAT ('  C) THERE IS NO RECORD OF THIS AUTHOR AND CATALOG NUMBER
     X')
      RETURN 1
C
C CHECK FOR RESERVATION
C
   85 IF (IREC(IX,31).EQ.0) GO TO 86
      IF (IREC(IX,31).EQ.LIBID)     GO TO 88
      XTRANS = .FALSE.
      GO TO 39
C
C MOVE UP THE RESERVATIONS
C
88    IREC(IX,31) = IREC(IX,32)
      IREC(IX,32) = IREC(IX,33)
      IREC(IX,33) = 0
86    IF(.NOT.XREQ)   GO TO 90
      WRITE (NOUT, 87)
   87 FORMAT ('  C) THIS BOOK IS AVAILABLE.')
      RETURN 1
C
C
C ENTER LOAN
C
   90 IREC(IX, 30) = LIBID
      IREC(IX, 35) = IREC(IX, 35) + 1
   92 IREC(IX, 34) = NDATE
C UPDATE USER'S RECORD
C     CALL LBOOK (LIBID)                          ***** REMOVED  ********
      J = IX
      GO TO 60
C
C ENTER RETURN
C
   95 IF (IREC(IX, 30).EQ.LIBID)     GO TO 97
      XTRANS = .FALSE.
      GO TO 66
C
97    XRES=.FALSE.
C CHECK FOR RESERVATION
      IF(IREC(IX,31).NE.0) XRES=.TRUE.
C CHECK FOR RENEWAL
      IF(.NOT.XRENEW) GO TO 110
      IF(.NOT.XRES) GO TO 110
C
C CHECK TIME REMAINING  FOR LOAN
      LEFT = IREC(IX,34) + IREC(1,37) - NDATE
      IF(LEFT.GT.0)  GO TO 124
      WRITE(6,101)
101   FORMAT('  C) CANNOT RENEW THIS BOOK AS IT HAS BEEN RESERVED AND YO
     CUR TIME IS UP.')
```

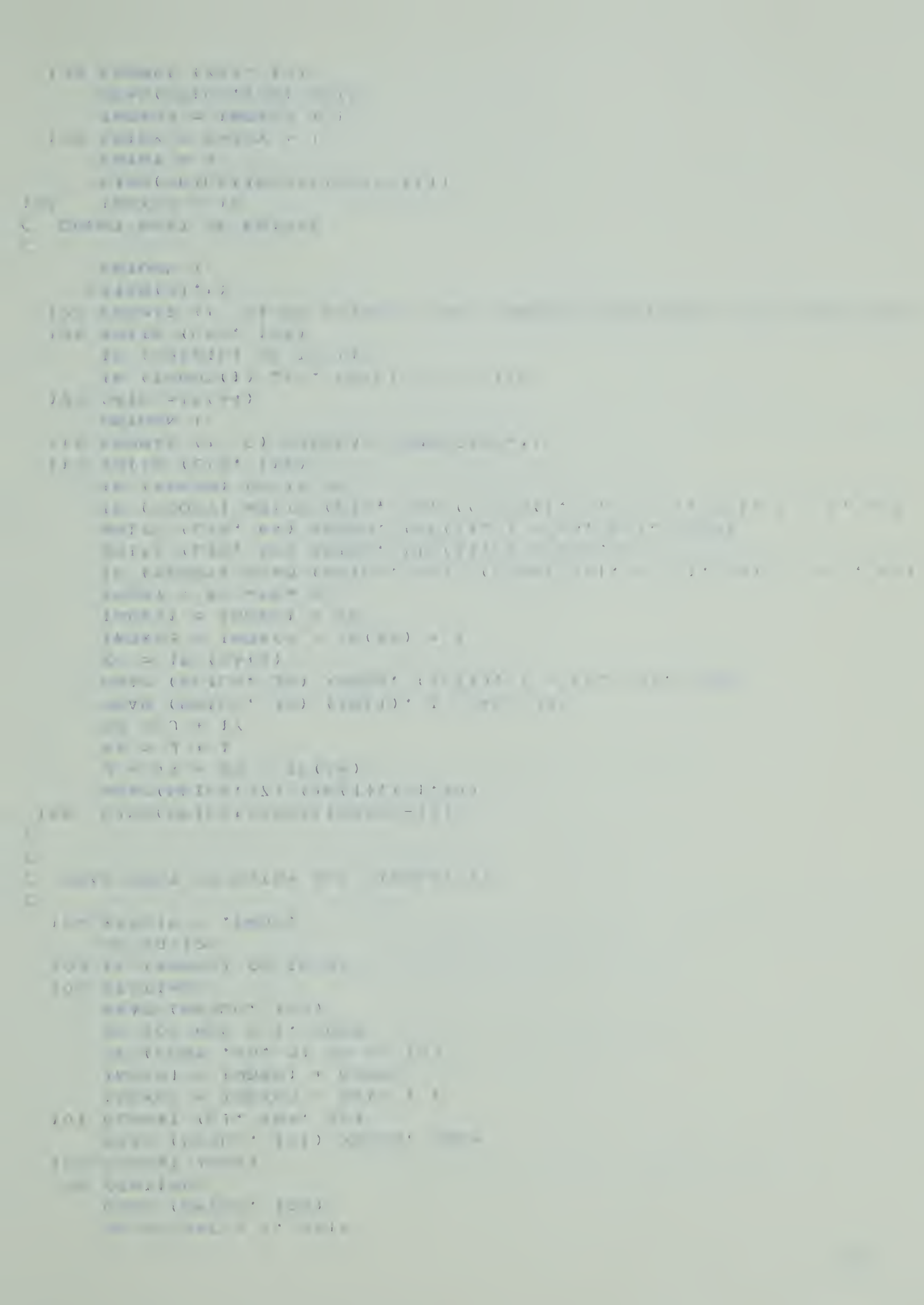
```
      GO TO 126
124   WRITE(6,125) LEFT
125   FORMAT('  C) THIS BOOK HAS BEEN RESERVED, BUT YOU HAVE '
     C,I2,' DAYS LEFT.'/5X,'DO YOU WISH TO KEEP THIS BOOK UNTIL IT IS DU
     CE?')
      READ (5,43,END=124) IREPLY
C UPDATE THE SYSTEM LOG
      WRITE(7,43) IREPLY
      IF(IREPLY.EQ.KYES)  GO TO 110
126   XRENEW = .FALSE.
C COMPLETE RETURN TRANSACTION
110   IREC(IX, 30) = 0
      IREC(IX,34) = NDATE
C
C UPDATE THE USER'S RECORD
C     CALL RBOOK(LIBID,CATLOG)                                  *** REMOVED *
      IF(XRENEW)   GO TO 90
C
C IF THERE IS A RESERVATION ON THIS COPY OF
C THE BOOK THEN INFORM THE LIBRARIAN.
      IF(XRES) WRITE(6,131) IREC(IX,31)
131   FORMAT('-C) LIBRARIAN:  HOLD THIS BOOK FOR USER NO.',I4//)
      J = IX
      GO TO 60
      END




      SUBROUTINE SAVE(*)
C
C THIS SUBROUTINE HANDLES THE ON-LINE COMMANDS
C 'SAVE', 'COPY', 'DISPLAY', AND 'DUMP' WHICH
C ARE HANDLED BY THE ENTRY POINTS 'SAVE',
C 'COPY', 'DISPLA', AND 'DUMP' RESPECTIVELY.
C
C MTS VERSION..... LAST UODATED ON MAY 9, 1970 BY J. DIMSDALE
C
C
      IMPLICIT INTEGER (A-W), LOGICAL(X-Z), REAL($)
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      DIMENSION DATE(3)
      COMMON/PARAM/DUMBY(10),INDX01,INDX02,INDX03,INDX04,INDX08
      INDX01 = 1
      INDX03 = 11
      FIND(1'1000*INDX01)
      FIND(3'1000*INDX03)
      REWIND TAPE
```

```
      CALL TRNSFR (NFILE, MFILE, TAPE, TAPE, &21)
   21 WRITE (LIB, 22)
   22 FORMAT ('  C) SAVE COMPLETED.')
      RETURN 1
C
C
      ENTRY COPY(*)
      INDX01 = 1
      INDX03 = 11
      FIND(1'1000*INDX01)
      FIND(3'1000*INDX03)
      REWIND TAPE
      CALL TRNSFR (TAPE, TAPE, NFILE, MFILE, &28)
   28 WRITE (LIB, 29)
   29 FORMAT ('  C) COPY COMPLETED.')
      RETURN 1
C
C
      ENTRY DUMP (*, ENTRY)
C
C
30    CALL DAY (NDATE, DATE)
      MCN = 7
      REWIND 7
      WRITE (MCN, 31) DATE
   31 FORMAT ('1 FILE DUMPED ', A4, I2, ' 19', I2, '.')
      ENTRY = 0
33    INDX01 = 1
      INDX03 = 11
      FIND(1'1000*INDX01)
      FIND(3'1000*INDX03)
   35 ENTRY = ENTRY + 1
      READ (NFILE, 37) (IF(I), I = 1, 44)
   37 FORMAT (24A1, 11I2, 8A4, I2)
      J = 44 + 80 * IF(44)
      KA = J + 1
      KB = J + 17
      READ (NFILE, 38) (IF(I), I = 45, J)
   38 FORMAT (80A1)
      READ (NFILE, 39) XNEND, (IF(I), I = KA, KB), $COST
   39 FORMAT (L1, 4A4, 4I2, 4I6, 3I2, I6, I2, F6.2)
      KC = IF(KA+5)
      YCOPY = KC .NE. 0
      IF (YCOPY) READ (MFILE, 40) ((LOAN(I,K),K=1,37),I=1,KC)
   40 FORMAT (24A1, I2, 4A4, 4I6, I4, I3, 2I2)
      WRITE (MON, 42) ENTRY, (IF(I), I = 1, J)
   42 FORMAT ('0ENTRY', I4, 5X, 24A1, 11I2, 8A4, I2 / (15X, 80A1))
      WRITE (MON, 44) XNEND, (IF(I), I = KA, KB), $COST
   44 FORMAT (13X, L1, 1X, 4A4, 4I2, 4I6, 3I2, I6, I2, F6.2)
      IF (YCOPY) WRITE (MON, 46) ((LOAN(I,K),K=1,37),I=1,KC)
   46 FORMAT (11X, 24A1, I2, 4A4, 4I6, I4, I3, 2I2)
   47 IF (XNEND) GO TO 35
   48 WRITE (LIB, 50)
   50 FORMAT ('0 C) DUMP COMPLETED.')
```

```
      RETURN 1
C
C
C  DISPLAY CAN BE USED TO DISPLAY PART OF THE CURRENT FILE
C
      ENTRY DISPLA (*, ENTRY)
C
   68 IF (INPUT(3) .LT. -300000000) GO TO 76
      WRITE (LIB, 69)
   69 FORMAT ('  C) INVALID DISPLAY PARAMETER(S).')
   70 WRITE (LIB, 71)
   71 FORMAT (/' TYPE:    IN ORDER TO:' /
     X 3X, '1     SPECIFY NEW DISPLAY PARAMETER(S)' /
     X 3X, '2     RECEIVE LINE PRINTER DUMP' /
     X 3X, '3     RETURN TO CALLING POINT WITHOUT DISPLAY OR DUMP')
      READ (LIBIN, 72) NUMBER
   72 FORMAT (I1)
      IF (NUMBER.LT.1 .OR. NUMBER.GT.3) NUMBER = 4
      GO TO (73, 30, 75, 77), NUMBER
   73 WRITE (LIB, 74)
   74 FORMAT ('  C) SPECIFY PARAMETER(S)')
      READ (LIBIN, 200) INPUT(3), INPUT(4), INPUT(5)
200   FORMAT(A4,A1,A4)
      GO TO 68
   75 IF (INPUT(1) - LIST(9)) 115, 134, 115
   77 WRITE (LIB, 78)
   78 FORMAT ('  C) INVALID RESPONSE')
      GO TO 70
C
C  IS INPUT (4) A HYPHEN?
   76 IF (INPUT(4) .NE. 1614823488) INPUT(5) = INPUT(3)
      IF (INPUT(5) .GT. -300000000) INPUT(5) = INPUT(3)
C
C  CHECK FOR SPAN OF FILE TO BE OUTPUT.
C  DISPLAY OF MORE THAN ABOUT 5 LETTERS NOT ALLOWED
      IF (INPUT(5)-INPUT(3)-200000000) 95, 95, 85
   85 WRITE (LIB, 86) INPUT(3), INPUT(5)
   86 FORMAT ('  C) DISPLAY EXCESSIVE (',A1,'AA TO ',A1,'ZZ) FOR TYPEWRI
     XTER.')
      GO TO 70
95    INDX01 = 1
      INDX03 = 11
      FIND(1'1000*INDX01)
      FIND(3'1000*INDX03)
      ENTRY = 0
      XBEGIN = .FALSE.
   96 ENTRY = ENTRY + 1
      READ (NFILE, 201) IF(1),IF(44)
201   FORMAT(A4,74X,I2)
      INDX03 = INDX03 + 1
      IF (INPUT(5) .LT. IF(1)) GO TO 120
      IF (XBEGIN) GO TO 106
      IF (INPUT(3) .LE. IF(1)) GO TO 104
      SKIP = IF(44)
```

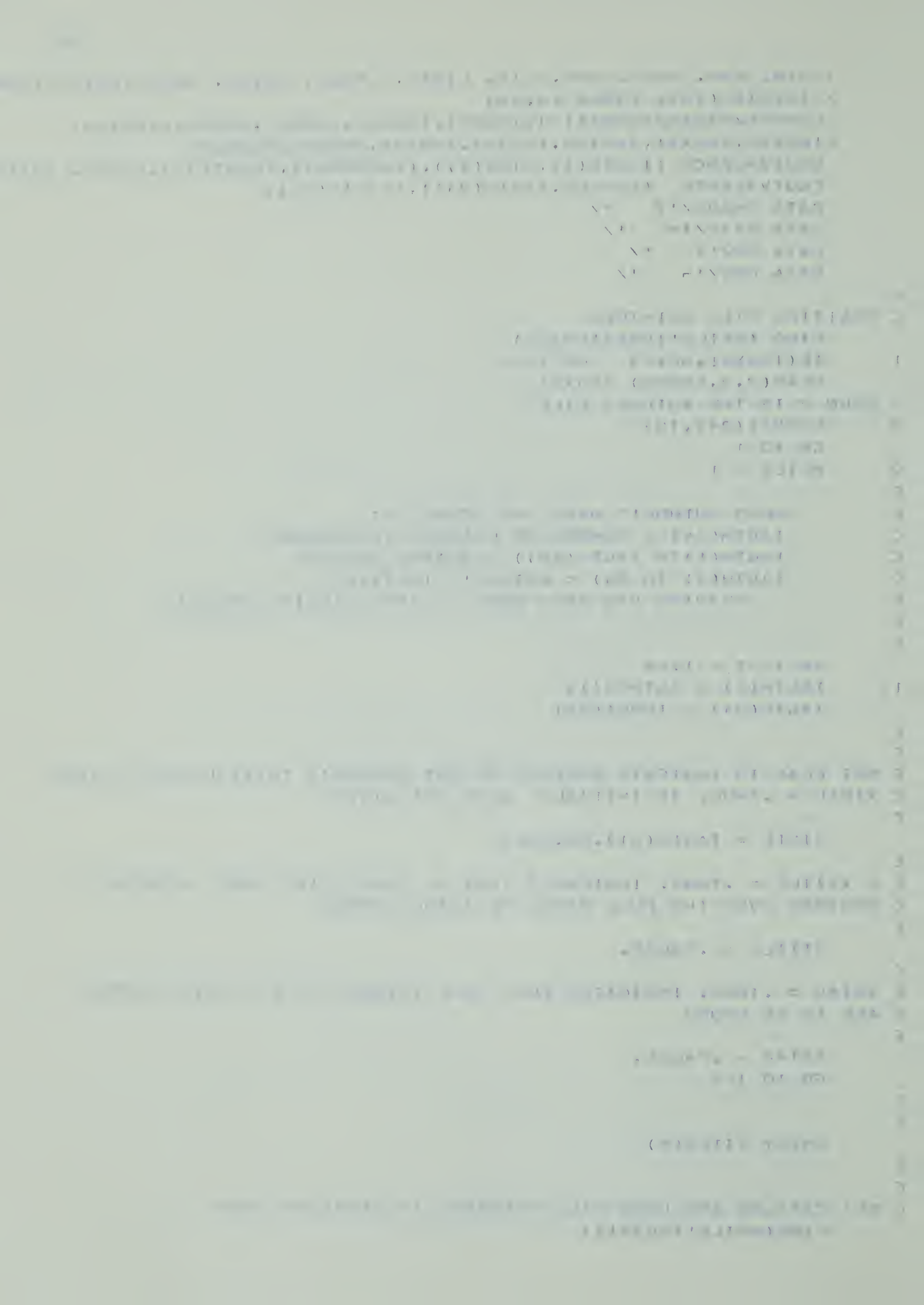


```
      DO 98 REC = 1, SKIP
      READ (NFILE, 100)
   98 CONTINUE
  100 FORMAT (80X)
      READ (NFILE, 101) XNEND, COPY
  101 FORMAT (L1, 18X, I2)
      INDX03 = INDX03 + SKIP + 1
      INDX01 = INDX01 + COPY
      IF (COPY .EQ. 0) GO TO 103
      DO 102 REC = 1, COPY
      READ (MFILE, 100)
  102 CONTINUE
  103 IF (XNEND) GO TO 96
      GO TO 125
  104 XBEGIN = .TRUE.
C
C  READ REST OF ENTRY AND DISPLAY IT
C
C
 106  FIND(NFILE'1000*(INDX03-1))
      READ(NFILE,37) (IF(I),I=1,44)
      J = 44 + 80 * IF(44)
      KA = J + 1
      KB = J + 17
      READ (NFILE, 38) (IF(I), I = 45, J)
      READ (NFILE, 39) XNEND, (IF(I), I = KA, KB), $COST
      KC = IF (KA+5)
      INDX03 = INDX03 + IF(44) + 1
      INDX01 = INDX01 + KC
      YCOPY = KC .NE. 0
      IF (YCOPY) READ (MFILE, 40) ((LOAN(I,K), K = 1, 37), I = 1, KC)
      WRITE (LIB, 42) ENTRY, (IF(I), I = 1, J)
      WRITE (LIB, 44) XNEND, (IF(I), I = KA, KB), $COST
      IF (YCOPY) WRITE (LIB, 46) ((LOAN(I,K), K = 1, 37), I = 1, KC)
      IF (XNEND) GO TO 96
  115 WRITE (LIB, 116)
  116 FORMAT ('  C) DISPLAY COMPLETED.')
      RETURN 1
  120 SKIP =IF(44)
      IF (INPUT(1) .EQ. KDEL) GO TO 140
      IF (XBEGIN) GO TO 115
  125 WRITE (LIB, 126)
  126 FORMAT ('  C) NO AUTHORS HAVE SURNAMES BEGINNING WITH SPECIFIED LE
     XTTER(S).')
      RETURN 1
C
C  COUNT REST OF ENTRYS
134   INDX03 = 10
      FIND(NFILE'1000*(INDX03+1))
      ENTRY = 0
  135 ENTRY = ENTRY + 1
      INDX03 = INDX03 + 1
      READ(NFILE,138) SKIP
  138 FORMAT (78X, I2)
```



```
140    INDX03 = INDX03 + SKIP + 1
       FIND(NFILE'1000*INDX03)
       READ (NFILE, 101) XNEND
       IF (XNEND) GO TO 135
       GO TO 115
       END



       SUBROUTINE SAVE(*)
C     OBJECT FORM OF THIS PROGRAM IS STORED IN DISK FILE "SAVE.BATCH"
C
C VERSION OF SAVE FOR  WHEN ADDING NEW BOOKS TO THE CATALOG AND LOAN
C FILES UNDER MTS BATCH
C        OBJECT MODULE ON DISK HAS THE NAME 'SAVE:BATCH'
C     LAST UPDATED ON MAY 12, 1970 BY J. DIMSDALE
       IMPLICIT INTEGER (A-W), LOGICAL(X-Z), REAL($)
       COMMON XTRACE, INDEX, NDATE,
      X IF(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
      X /COMAND/ LIST(22), KBLA, KYES,
      X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
      X ISTATU (12), LIBNM (4,10)
       COMMON/PARAM/DUMBY(10),INDX01,INDX02,INDX03,INDX04,INDX08
       FIND(MFILE'1000*INDX01)
       FIND(NFILE'1000*INDX03)
       REWIND TAPE
C TAPE IS NATURALLY ASSUMED TO BE A SEQUENTIAL TYPE FILE
       CALL TRNSFR (NFILE, MFILE, TAPE, TAPE, &21)
   21  WRITE (LIB, 22)
   22  FORMAT ('  C) SAVE COMPLETED.')
       RETURN 1
       END



       SUBROUTINE SEARCH (*)
C MTS VERSION.... LAST UPDATED ON  OCT. 4, 1970 BY J. DIMSDALE
C
C THIS SUBROUTINE HANDLES THE COMMAND 'SEARCH'.
C
       IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
       DIMENSION INP(80), IAUTH(34),                NSTORE(9,824)
       DIMENSION QUERY(4,4,21),WORDS(4)
       DIMENSION AUTHOR(24)
       LOGICAL TERM(4)
       COMMON XTRACE, INDEX, NDATE,
      X IF(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
      X /COMAND/ LIST(22), KBLA, KYES,
```

```
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE,OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      COMMON/PARAM/DUMMY(10),INDX01,INDX02,INDX03,INDX04,INDX08,
     CINDX20,INDX21,INDX22,INDX23,INDX24,INDX25,MAXAUT
      EQUIVALENCE (IAUTH(1),LINK(1)),(AUTHOR(1),INPUT(3)),(COPY,LAST(6))
      EQUIVALENCE  (NUMSUR,IAUTH(34)),(K3,IF(26))
      DATA QMARK/'?   '/
      DATA STAR/'*   '/
      DATA OR/'|   '/
      DATA NOT/'¬   '/
C
C POSITION FILE POINTERS
      FIND (NFILE'1000*INDX03)
1     IF(INDX01.NE.0)   GO TO 9
      READ(2,2,END=9) INDX01
C DSRN 2 IS THE AUTHORS FILE
2     FORMAT(24X,I5)
      GO TO 1
9     MFILE = 1
C
C         READ AUTHOR'S NAME AND STORE AS:
C         IAUTH(34) = NUMBER OF LETTERS IN SURNAME
C         IAUTH(1)TO IAUTH(20)) = AUTHOR SURNAME
C         IAUTH(21 TO 24) = AUTHOR'S INITIALS
C            READING HAS BEEN DONE IN THE MAINLINE PROGRAM
C
C
      DO 10 I = 1,24
10    IAUTH(I) = AUTHOR(I)
      IAUTH(34) = INPUT(32)
C
C
C SET FLAG TO INDICATE WHETHER OR NOT AUTHOR'S INITIALS WERE GIVEN
C XINTL = .TRUE. IF INITIALS  WERE NOT GIVEN
C
      XINTL = IAUTH(21).EQ.KBLA
C
C C XTITLE = .TRUE. INDICATES THAT A  PURE TITLE WORD  SEARCH IS
C DESIRED OVER THE FULL RANGE OF AUTHOR NAMES
C
      XTITLE = .FALSE.
C
C XSTAR = .TRUE. INDICATES THAT  ALL TITLES FOR THE GIVEN AUTHOR
C ARE TO BE FOUND
C
      XSTAR = .FALSE.
      GO TO 102
C
C
      ENTRY TITLE(*)
C
C
C SET CATALOG AND LOAN FILE POINTERS TO START OF FILE
      FIND(NFILE'1000*11)
```



```
      INDX01 = 1
      MFILE=1
      FIND(MFILE*1000+INDX01)
C
C SET FLAG TO INDICATE THAT SEARCH IS TO BE ON TITLE WORDS ONLY
      XTITLE = .TRUE.
      GO TO 102
  100 WRITE (6, 101)
101   FORMAT(' C) TYPE UP TO FOUR WORDS OF TITLE.  PROGRAM MATCHES  ON
     1TRUNCATED AS WELL AS COMPLETE WORDS.'/'    TYPING ''*'' INDICATES
     2THAT  YOU WANT ALL TITLES  FOR THE GIVEN AUTHOR.')
C
102   READ(5,16,END=100) INP
      WRITE(7,16) INP
16    FORMAT(80A1)
C
C DOES THE USER REQUIRE PROMPTING?
      IF(INP(1).EQ.QMARK) GO TO 100
C
C
C QUERY INPUT SECTION
C
C
C TITLE QUERIES MAY CONTAIN UP TO 4 TERMS EACH OF WHICH MAY CONSIST
C OF 4 WORDS IN 'OR' RELATIONSHIP.      'AND' LOGIC  APLLIES TO THE
C TERMS.
C
C  QUERY(T,W,L) = L TH LETTER OF W TH WORD IN T TH TERM OF INPUT LINE
C    QUERY(T,W,21) = NUMBER OF LETTERS IN W TH WORD OF T TERM
C WORDS(T) = NUMBER OF WORDS IN T TH TERM OF INPUT LINE
C
C
C  T = TERMS-IN-QUERY COUNTER
      T=1
C W = WORDS-IN-TERM COUNTER
      W=1
C L= LETTERS-IN-WORD COUNTER
      L=0
C INITIALIZE THE TERM LOGIC VECTOR.
      DO 110 I=1,4
110   TERM(I)=.TRUE.
C
C SCAN THE INPUT LINE
      DO 130 I=1,80
      IF(INP(I).EQ.STAR) GO TO 190
      IF(INP(I).EQ.KBLA) GO TO 119
      IF(INP(I).EQ.OR)  GO TO 120
      IF(INP(I).EQ.NOT) GO TO 128
C FOUND A NEW LETTER
      L=L+1
      QUERY(T,W,L) = INP(I)
      IF(L.GT.20) GO TO 140
      GO TO 130
C
```

```
C END OF CURRENT WORD
119    XOR = .FALSE.
       GO TO 121
120    XOR = .TRUE.
121    QUERY(T,W,21) = L
       WORDS(T) = W
       IF(XOR)  GO TO 125
C END OF QUERY?   INDICATED BY  TWO BLANKS IN SUCCESSION.
       IF(INP(I+1).EQ.KBLA)  GO TO 150
C END OF TERM
       T=T+1
       W=0
       IF(T.GT.4) GO TO 142
125    W=W+1
       IF(W.GT.4) GO TO 144
       L=0
       GO TO 130
128    TERM(T)=.FALSE.
130    CONTINUE
C
C
C ERROR MESSAGES
C
140    WRITE(6,141)
141    FORMAT(' C) NO MORE THAN 19 LETTERS PER WORD.')
       GO TO 102
142    WRITE(6,143)
143    FORMAT(' C) NO MORE THAN 4 TERMS PER QUERY.')
       GO TO 102
144    WRITE(6,145)
145    FORMAT(' C) NO MORE THAN 4 WORDS PER TERM.')
       GO TO 102
C
150    TERMS = T
       IF(XTRACE) GO TO 200
       DO 158 T=1,TERMS
       WRITE(6,157)
157    FORMAT(' .AND.')
       W2=WORDS(T)
       DO 156 W=1,W2
       L2 = QUERY(T,W,21)
       WRITE(6,155) L2,(QUERY(T,W,L),L=1,L2)
155    FORMAT(' .OR. ',I5,20A1)
156    CONTINUE
158    CONTINUE
       GO TO 200
C
190    XSTAR =.TRUE.
C
C CHECK FOR CASE OF BOTH AUTHOR AND TITLE BEING UNSPECIFIED
C
       IF(XTITLE.AND.XSTAR ) GO TO 700
C
200    XAUTH = .FALSE.
```

```
      XINIT = .FALSE.
C
C        INDEX COUNTS NO. OF CORRECT ENTRYS OUTPUT (MORE OR LESS)
      INDEX = 0
C
C RESET THE HITS INDICATOR
      XNFILE = .FALSE.
      INSTOR = 0
C
C
C
C        READ ONE ENTRY FROM FILE (NFILE)
C        FILE STRUCTURE FOR EACH INDEX CARD IS:
C    RECORD 1:  IF(1-25)  = AUTHOR SURNAME AND INITIALS AS IN IAUTH(1-25
C                           IN FORMAT (24A1, I2)
C               IF(26)    = NUMBER OF TITLE WORDS, FORMAT (I2)
C               IF(27-35) = POSITIONS OF 1ST LETTERS OF WORDS, IF(27)=1,
C               IF(36-43) = AUTHOR DETAILS, FORMAT (8A4)
C               IF(44)    = NUMBER OF RECORDS FOLLOWING IN 80A1, FORMAT
C    RECORDS:   OF NUMBER IF(44), FORMAT (80A1)
C    RECORD:    END FLAG (F OR T), CAT NO., LIBRARY NUMBER, NUMBER OF CO
C               LOAN PERIOD (DAYS), NUMBER OF OTHER AUTHORS, AS MANY AS
C               4 BOOK NUMBERS, DATE RECEIVED, ORDER NUMBER, FUND NO., C
C               FORMAT (L1, 4A4, 4I2, 4I7, 3I2, I6, I2, F6.2)
C
C
C
      COPY =0
C
C  UPDATE LOANS FILE POINTER SO THAT IT POINTS TO FIRST LOANS RECORD
C FOR THE CATALOG ENTRY ABOUT TO BE READ
C COPY = NUMBER OF COPIES OF BOOK HELD BY LIBRARY.
C THERE IS ONE LOANS FILE RECORD FOR EACH COPY OF EACH BOOK
C
210   INDX01 = INDX01 + COPY
C
C READ CATALOG ENTRY
      READ (NFILE, 211,END=510  ) (IF(I), I = 1, 44)
  211 FORMAT (24A1, 11I2, 8A4, I2)
      J = 44 + ((80*IF(44))/66)*66
      READ(NFILE, 213,END=  510)  (IF(I), I=45,J)
  213 FORMAT (80A1)
      IF (IF(J-63) .EQ. KBLA) J = J - 66
      READ(NFILE, 218 ,END=  510) XNEND,(LAST(I),I=1,17), $COST
  218 FORMAT (L1, 4A4, 4I2, 4I6, 3I2, I6, I2, F6.2)
C
C SET XMAIN = .TRUE. IF CATALOG ENTRY IS FOR MAIN AUTHOR
      XMAIN = COPY.NE.0
C
C XSEC= .TRUE.  IF CATALOG ENTRY IS FOR SECONDARY AUTHOR
      XSEC = .NOT.XMAIN
C
C
C
```

```
C
C  IF A TITLE MATCH ONLY IS REQUIRED THEN SKIP THE AUTHOR MATCH
      IF(XTITLE) GO TO 250
C
C         SET XAUTH = T IF CORRECT AUTHOR SURNAME FOUND
225   DO 230 I = 1,NUMSUR
      IF(IAUTH(I) - IF(I))  510,230,210
  230 CONTINUE
      XAUTH = .TRUE.
C
C IF SURNAME MATCH ONLY IS REQUIRED THEN SKIP INITIALS MATCH
      IF(XINTL)  GO TO 249
C
C         SET XINIT = T IF CORRECT AUTHOR INITIALS
      XINIT = .FALSE.
      DO 240 I = 21, 24
      IF (IAUTH(I).NE.IF(I)) GO TO 210
  240 CONTINUE
      XINIT = .TRUE.
C
C IF AUTHOR ONLY MATCH IS REQUIRED THEN SKIP THE TITLE MATCH
249   IF(XSTAR) GO TO 309
      GO TO 251
C
C  MAIN AUTHOR ENTRY?  IF NOT THEN GO READ THE NEXT ENTRY IN THE FILE
250   IF(XSEC) GO TO 210
C
C         SET XTITL = T IF CORRECT TITLE WORDS FOUND
C
C
C
C TITLE MATCH SECTION FOR 'AND-OR' LOGIC  TYPE SEARCH QUESTIONS
C SET XTITL= T IF CORRECT COMBINATION OF QUERY TERMS FOUND.
C N.B. EACH TERM MAY CONSIST OF UP TO FOUR WORDS IN AN 'OR'
C RELATIONSHIP.
C  IF SUPPLIED WORD IN QUERY IS A TRUNCATED FORM OF A TITLE WORD,
C THEN IT WILL MATCH.
C
251   XTITL= .FALSE.
C
C  IF QUERY CONTAINS MORE TERMS THAN  THERE ARE WORDS IN THE  ENTRY'S
C  TITLE, THEN NO HIT IS POSSIBLE FOR THIS ENTRY.
      IF(TERMS.GT.K3)   GO TO 350
C TERM LOOP
      DO 290 T=1,TERMS
      W2 = WORDS(T)
C WORDS-IN-TERM LOOP
      DO 280 W=1,W2
      L2 = QUERY(T,W,21)
C WORDS-IN-TITLE LOOP
      DO 280 I=1,K3
      M=IF(I+26)+43
C LETTERS-IN-WORD LOOP
      DO 260 L=1,L2
```

```
      IF(QUERY(T,W,L).NE.IF(M+L))  GO TO 280
260   CONTINUE
C CHECK FOR 'NOT' LOGIC.
      IF(TERM(T)) GO TO 290
      GO TO 350
280   CONTINUE
C IF NO HITS IN TERM THEN THIS TITLE
C CANNOT BE A HIT FOR THE QUERY.
C CHECK FOR 'NOT' LOGIC.
      IF(TERM(T)) GO TO 350
C GOT HIT IN TERM,  SO TRY NEXT TERM
290   CONTINUE
C WE HAVE A HIT FOR THE QUERY,  SO OUTPUT IT
      XTITL = .TRUE.
C
C
C  HAVE FOUND CORRECT COMBINATION OF SURNAME, INITIALS, AND TITLE WORDS
C   AS SPECIFIED IN THE SEARCH COMMAND
C        OUTPUT ENTRY
C
309   CONTINUE
      WRITE (6, 310) (IF(IA), IA = 36, 43), (IF(IB), IB = 45, J)
  310 FORMAT (/ '  C) ', 8A4 / (8X, 66A1))
      IC = 10
C
C IF CATALOG ENTRY IS FOR MAIN AUTHOR THEN READ THE CORRESPONDING
C LOANS FILE RECORD(S).
      IF(XSEC) GO TO 312
      FIND(MFILE'1000*INDX01)
      READ(MFILE,220,END=510)((LOAN(I,K),K=1,4),I=1,COPY)
  220 FORMAT (42X, 4I6)
      CALL STATUS(IC,COPY)
312   ID = IC + 2
      WRITE (6,311) (LAST(I), I = 1, 4),(LIBNM(I,LAST(5)), I = 1, 4),
     X (ISTATU(I), I = IC, ID)
  311 FORMAT (10X, 'CATALOG NO. ', 4A4, ' IN ', 4A4, ' LIBRARY' / 10X,
     X 'STATUS:  ', 3A4)
      IF(.NOT.XTRACE) WRITE(6,220)INDX01
C SET HITS INDICATOR
      XNFILE = .TRUE.
      IF (COPY .EQ. 0) INDEX = 2
      INDEX = INDEX + 1
      IF (INDEX.NE.1) GO TO 500
C
C
C     THE VECTOR "LINK" CONTAINS INFORMATION REQUIRED BY OTHER SUBROUTINE
C     FROM THE FIRST CORRECT ENTRY FOUND.
C     LINK(1-24)     AUTHOR'S NAME
C     LINK(25-28)    CATALOG NUMBER
C     LINK(29-32)    NAME OF LIBRARY HOLDING BOOK
C     LINK(33)       1 IF THIS BOOK AVAILABLE, 4 IF REQUESTS STILL ACCEPTE
C                    7 IF NO MORE REQUESTS
C     LINK(34)       LENGTH OF AUTHOR SURNAME
C
```

```
C
      DO 320 IA = 25, 28
      LINK(IA) = LAST(IA-24)
      LINK (IA+4) = LIBNM (IA-24, LAST(5))
  320 CONTINUE
      LINK(33) = IC
      GO TO 500
C
C         IF INCORRECT TITLE WORDS BUT CORRECT SURNAME AND CORRECT INITIA
C         PLACE IN STORE IF XNFILE = .FALSE. AND INCREMENT INSTOR.
C         IF INCORRECT INITIALS GO TO 500
C
C  NSTORE(INSTOR,1) = NO. OF WORDS IN TEXT TO BE OUTPUT
C  NSTORE(INSTOR,2 TO 9)                          = IF(36 TO 43)  AUTHOR DET
C  NSTORE(INSTOR,10 TO END OF TEXT) = IF(45 TO END)
C  NSTORE(INSTOR,END+1 TO END + 4) = LAST(1 TO 4)
C  NSTORE(INSTOR,END+5) = LAST(5)
C  NSTORE(INSTOR,END+6) = IC
C
C
C TITLE MATCH NOT FOUND
C IF TITLE MATCH ONLY IS REQUIRED THEN GO TRY NEXT ENTRY
350   IF(XTITLE)  GO TO 210
      IF (XNFILE .OR. .NOT. XINIT) GO TO 500
C
C
C
C
      INSTOR = INSTOR + 1
      NSTORE(INSTOR,1) = J - 35
      DO 355 IG = 2, 9
      NSTORE(INSTOR,IG) = IF(34+IG)
  355 CONTINUE
      IH = J - 35
      DO 360 IG = 10, IH
      NSTORE(INSTOR,IG) = IF(IG+35)
  360 CONTINUE
      DO 365 IG = 1, 4
      NSTORE(INSTOR,IH+IG) = LAST(IG)
  365 CONTINUE
      NSTORE(INSTOR,IH+5) = LAST(5)
      IC = 10
      IF(XSEC) GO TO 390
      FIND(MFILE'1000*INDX01)
      READ(MFILE,220,END=510)((LOAN(I,K),K=1,4),I=1,COPY)
      CALL STATUS(IC,COPY)
  390 NSTORE(INSTOR,IH+6) = IC
  395 IF (INSTOR .LT. 9) GO TO 500
      WRITE (LIB, 400) (IAUTH(I), I = 1, 24)
  400 FORMAT ('  C) VECTOR "NSTORE" FILLED BY 9 ENTRYS OF AUTHOR ',24A1)
      GO TO 510
C
C         IF NOT END OF FILE EXAMINE NEXT ENTRY
C
```

```
C IF NOT LAST ENTRY IN CATALOG FILE THEN GO READ ANOTHER
  500 IF (XNEND) GO TO 210
C
C         REPLY TO USER IF NECESSARY
C IF HITS WERE FOUND AND PRINTED THEN RETURN
  510 IF (XNFILE) RETURN 1
C WAS THIS A TITLE-WORDS-ONLY SEARCH?
      IF(XTITLE) GO TO 530
C     DID WE SAVE ANY AUTHOR MATCHES  FOR THIS SEARCH?
C IF YES THEN PRINT THEM
      IF(INSTOR.EQ.0) GO TO 550
      WRITE(6,515) INSTOR
515   FORMAT('  C) THIS AUTHOR WAS FOUND BUT NOT WITH THE GIVEN '
     1'COMBINATION OF TITLE WORDS.'/5X,'WE FOUND ',I3,' ENTRIES '
     2'FOR THIS AUTHOR.'/5X,'TYPE YES IF YOU WISH THEM DISPLAYED;'
     3' OTHERWISE TYPE NO.')
      READ(5,517) ANSWER
517   FORMAT(A4)
      IF(ANSWER.NE.KYES) RETURN 1
      DO 520 IK = 1, INSTOR
      IH = NSTORE(IK,1)
      WRITE (6, 310) (NSTORE(IK, IM), IM = 2, IH)
      IQ = IH + 1
      IR = IH + 4
      IC = NSTORE(IK,IH+6)
      ID = IC + 2
      WRITE (6,311) (NSTORE(IK,IM), IM = IQ, IR),
     X (LIBNM(IM,NSTORE(IK,IH+5)), IM = 1, 4), (ISTATU(IM), IM = IC, ID)
  520 CONTINUE
      RETURN 1
C
530   WRITE(6,531)
531   FORMAT('  C) NO HITS FOR THE GIVEN COMBINATION OF TITLE WORDS.')
      RETURN 1
C
C  DID WE FIND A MATCH ON SURNAME?
  550 IF (.NOT. XAUTH) GO TO 600
C
C WERE INITIALS SUPPLIED?
      IF(XINTL) GO TO 560
C
      WRITE (6, 555)
  555 FORMAT ('  C) THIS SURNAME WAS FOUND BUT NOT WITH THE GIVEN INITIA
     XLS OR TITLE WORDS.')
556   RETURN 1
560   WRITE(6,561)
561   FORMAT('  C) THIS SURNAME WAS FOUND BUT NOT WITH THE'/
     15X,'GIVEN COMBINATION OF TITLE WORDS.')
      RETURN 1
  600 WRITE (6, 601)
  601 FORMAT ('  C) THERE IS NO RECORD OF THIS AUTHOR')
      RETURN 1
700   WRITE(6,701)
701   FORMAT(' C) YOU MUST SPECIFY  AUTHOR OR TITLE WORDS.')
```

```
      RETURN 1
      END



      SUBROUTINE SETUP(*)
C
C  MTS VERSION....LAST UPDATED ON OCT. 21, 1970 BY J. DIMSDALE
C
      IMPLICIT INTEGER(A-W),LOGICAL(X-Z),REAL($)
      INTEGER XNAME,XIDNUM
      DIMENSION NAMKEY(500,6),IDNKEY(500,2),ADDR(6),NAME(5),DEPT(2),
     CPHON(2),FSL(19),DEL(5),CON(5),XNAME(5)
      COMMON /TABLES/NAMKEY,IDNKEY,FSL,DEL,CON,FSL24(26)
      COMMON /PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL,USTOR1,OSTOR1,FSIN24,
     CALFLAG,MAXPTR,INDX01,INDX02,INDX03,
     CINDX04,INDX08,INDX20,INDX21,INDX22,INDX23,INDX24,FLOOR
      COMMON/USER/XNAME,STATUS,TITLE,ADDR,PHON,DEPT,XIDNUM,K1,K2,ACCT,
     CENDATE,ONLOAN,OVRDUE,OVER1,DUMMY
      EQUIVALENCE (USTOR1,USTOR),(OSTOR1,OSTOR)
      DATA IMAX/500/,JMAX/6/,KMAX/2/,WA/5/,WT/6/
C
C SETUP INITIALIZES THE IN-CORE USER TABLES AND ASSOCIATED LISTS AND
C PARAMETERS.
C
C
C THE FOLLOWING CORRESPONDENCE HOLDS BETWEEN
C DSRN'S AND FILE NAMES:
C        4 = USERS
C        8 = USERINDEX
C        9 = FREESPACE
C N.B. UPPER PORTION OF FILE FREESPACE = FILE FREESTOR
C AND LOWER PORTION = FILE FREELOCS.
C
C READ FIRST RECORD FROM FILE USERS.     IT CONTAINS THE FOLLOWING FILE
C PARAMETERS:
C            TOTAL = NUMBER OF  RECORDS IN FILE USERINDEX
C                  = NUMBER OF USERS CURRENTLY ENROLLED
C            MAXID = POINTER TO THE LAST RECORD
C                    IN FILE USERS
C            USTOR1 = POINTER TO THE LAST FREE STORAGE
C                     LIST IN FILE FREESTOR
C            FLOOR = POINTER TO THE FIRST FREE STORAGE
C                    LIST IN FILE FREELOCS
C            OSTOR1 = POINTER TO THE LAST FREE STORAGE
C                     LIST IN FILE FREELOCS
C            MAXPTR = POINTER TO THE LAST RECORD IN FILE
C                     OVERDUES
C
      FIND(4'1000)
      READ( 4,1,END=250) TOTAL,MAXID,USTOR1,OSTOR1,MAXPTR,FLOOR
```

```
    1 FORMAT(20I4)
      IF( TOTAL.GT.0) GO TO 280
C
250   WRITE(6,251)
251   FORMAT(' *****  THE USER DIRECTORY DOES NOT EXIST      ***')
      TOTAL=0
      MAXID=1
      GO TO 282
C
C
C READ THE NAME AND ID NUMBER INDEX TABLES
C INTO CORE FROM FILE USERINDEX
C
280   FIND(8'1000)
      DO 3 J=1,TOTAL
      READ(8,2) (NAMKEY(J,K),K=1,6),(IDNKEY(J,K),K=1,2)
 2    FORMAT(5A4,1X,I4,1X,I9,1X,I4)
 3    CONTINUE
C
C IF NEED BE, INITIALIZE  FILE PARAMETERS
C
282   IF(USTOR.EQ.0) USTOR=1
      IF(FLOOR.EQ.0) FLOOR=2
      IF(OSTOR.EQ.0) OSTOR=FLOOR
C
C
C READ THE LAST RECORD FROM FILE FREESTOR INTO THE
C IN-CORE FREE STORAGE LIST FOR FILE USERS
C
      FIND(9'1000*USTOR)
      READ(9,  1) FSINDX,FSL
C
C READ THE LAST RECORD FROM FILE FREELOCS INTO THE
C IN-CORE FREE STORAGE LIST FOR FILE OVERDUES
C
      FIND(9'1000*OSTOR)
      READ(9,289) FSIN24,FSL24
289   FORMAT(I2,26I3)
      RETURN 1
C
C
C
      ENTRY LOOK1(NAME,LIBID,SEE,*,*)
C
C TRANSLATE USER NAME INTO USER INTERNAL ID
C NUMBER BY PERFORMING A BINARY  SEARCH
C TABLE LOOKUP ON USER NAME INDEX
C
      SWITCH=1
      FILE=6
      CALL BINS1(NAMKEY,IMAX,JMAX,TOTAL,WT,NAME,WA,LIBID,K1,&300,&305)
300   INDX20 = LIBID
C
C VALUE OF SEE SPECIFIES WHETHER USER'S
```

```
C RECORD IS TO BE READ OR NOT, AND IF IT IS
C TO BE READ THEN WHETHER OR NOT IT SHOULD
C BE WRITTEN OUT TO THE TERMINAL OR TO
C SOME OTHER DEVICE
C
      IF(SEE) 301,302,302
301   RETURN 1
C
C READ USER'S RECORD FROM FILE USERS
C
302   FIND(4'1000*INDX20)
      READ( 4,303) XNAME,STATUS,TITLE,ADDR,PHON,DEPT,XIDNUM,ACCT,ENDATE,
     CONLOAN,OVRDUE,OVER1,DUMMY
303   FORMAT(5A4,2I1,7A4,A3,2A1,I9,I1,I4,2I2,I3,I4)
      IF(SEE) 298,298,297
C
C WRITE OUT USER'S RECORD ON TERMINAL OR
C ON OTHER DEVICE AS SPECIFIED BY VALUE
C OF FILE
C
297   WRITE(FILE,318)
      WRITE(FILE,304) XNAME,STATUS,TITLE,ADDR,PHON,DEPT,XIDNUM,ONLOAN,
     COVRDUE,LIBID
298   GO TO (301,308,312),SWITCH
304   FORMAT(1X,5A4,1X,I1,1X,I1,4X,7A4,A3,1X,2A1,8X,I9,7X,I2,13X,I2,
     C10X,I4)
305   WRITE(FILE,315) NAME
      WRITE(FILE,317)
      RETURN 2
C
C
      ENTRY LOOK2(IDNUM,LIBID,SEE,* ,*)
C
C TRANSLATE THE USER'S IDENTITY NUMBER INTO
C THE USER'S INTERNAL IDENTITY NUMBER BY
C PERFORMING A BINARY SEARCH TABLE LOOKUP
C ON THE USER ID NUMBER INDEX
C
      SWITCH=2
      FILE=6
      CALL BINS2(IDNKEY,IMAX,KMAX,TOTAL,IDNUM,LIBID,K2,&307,&309)
307   INDX20 = LIBID
      IF(SEE) 308,302,302
308   RETURN 1
309   WRITE(FILE,316) IDNUM
      WRITE(FILE,317)
      RETURN 2
C
C
      ENTRY LOOK3(LIBID,SEE,FILE,*,*)
C
C CHECK TO MAKE SURE THAT SUPPLIED INTERNAL
C USER IDENTITY NUMBER IS WITHIN THE
C CORRECT RANGE
```

```
C
      IF(LIBID.LT.2.OR.LIBID.GT.MAXID) GO TO 313
      SWITCH=3
      INDX20 = LIBID
      IF(SEE) 312,302,302
312   RETURN 1
313   WRITE(FILE,314) LIBID
314   FORMAT(1X,I10,'   IS AN INVALID LIBRARY ID NUMBER')
      RETURN 2
315   FORMAT(1H+,5A4)
316   FORMAT(1H+,I9)
317   FORMAT(1X,'   IS NOT IN THE USER DIRECTORY')
318   FORMAT(' SURNAME          INIT STATUS ADDRESS                    PHONE
     C   DEPT        IDNUMBER   BOOKS BORROWED BOOKS OVERDUE  LIBID')
C
C
      ENTRY PUT(*)
C
C PUT UPDATED USER RECORD BACK INTO FILE USERS
C
      FIND(4'1000*LIBID)
399   WRITE( 4,303) XNAME,STATUS,TITLE,ADDR,PHON,DEPT,XIDNUM,ACCT,ENDATE
     C,CNLOAN,OVRDUE,OVER1,DUMMY
401   RETURN 1
C
C
      ENTRY PUT2(XOLD)
C
C WRITE OUT UPDATED CONTENTS OF THE IN-CORE
C FREE STORAGE LIST FOR FILE USERS
C
      FIND(9'1000*USTOR)
      WRITE( 9,  1) FSINDX,FSL
C
C DID WE WRITE OUT A FULL FREE STOAGE LIST?
C IF YES, THEN INCREMENT THE POINTER FOR
C FILE 'FREESTOR'.  OTHERWISE RETURN.
C
      IF(XOLD) RETURN
C
C INCREMENT POINTER FOR FILE FREESTOR
      USTOR=USTOR+1
C
C SHOULD WE MOVE FILE FREELOCS?
      IF(USTOR.GE.FLOOR) GO TO 500
C
C WRITE OUT UPDATED FILE PARAMETERS
450   FIND(4'1000)
      WRITE( 4,1) TOTAL,MAXID,USTOR1,CSTOR1,MAXPTR,FLOOR
      RETURN
C
C SHIFT FILE FREELOCS DOWNWARD BY 5 LINES
C SO AS TO ALLOW FILE FREESTOR TO EXPAND
C
```

```
500     J=OSTOR-FLOOR+1
        K=OSTOR+1
        DO 510 I=1,J
        K=K-1
        FIND(9'1000*K)
        READ(9,505) RECORD
505     FORMAT(20A4)
        FIND(9'1000*(K+5))
510     WRITE(9,505) RECORD
        FLOOR=FLOOR+5
        OSTOR=OSTOR+5
        GO TO 450
        END



        SUBROUTINE SORT1
C
C THIS SUBROUTINE SORTS THE IN-CORE NAME INDEX TABLE
C IT ALSO CHECKS FOR AND DELETES ANY ATTEMPTED DUPLICATION OF AN
C EXISTING USER RECORD.  IT MAKES A LIST OF SUCH DUPLICATIONS ( DEL )
C AND A LIST OF THE ORIGINAL  RECORDS WHICH CONFLICTED WITH INCOMING
C RECORDS ( CON ).  BOTH ARRAYS,  DEL AND CON, ARE PASSED TO SUBROUTINE
C SORT2 ( WHICH SORTS THE IN-CORE IDNUMBER INDEX TABLE) AND TO ENROLL
C WHICH CALLS BOTH  SORT1 AND SORT2 IN TURN.
C SORTING IS DONE BY A MODIFIED 'SINKING' SORT.
C
C
        IMPLICIT INTEGER(A-W),LOGICAL(X-Z),REAL($)
        DIMENSION NAME(500,6),IDNUM(500,2), FSL(19),DEL(5),CON(5)
        COMMON /TABLES/NAME,IDNUM,FSL,DEL,CON
        COMMON /PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL
        COMMON /CROSS/DELET1
        DATA KZ/'ZZZZ'/, KBLA/'    '/
C
C
        DELET1=0
 105    I=TOTAL
1       J=0
        EXFLAG=0
2       J=J+1
        IF(J.GE.I) GO TO 8
        JJ=J+1
        DO 3 K=1,5
        IF (NAME(J,K)  -  NAME(JJ,K)) 4,3,6
3       CONTINUE
        GO TO 5
4       IF(NAME(JJ,K).EQ.KBLA) GO TO 6
        GO TO 2
C
C
```

```
C IF THE FIRST WORD OF A DUPLICATE NAME IS 'ZZZZ' THEN IT IS A DELETION
C CAUSED BY THE COMMAND 'REMOVE' OR BY THE NEXT SECTION OF THIS
C PROGRAM
C
5       IF(NAME(JJ,1).EQ.KZ) GO TO 2
C
C DELETE DUPLICATE RECORD AND STORE ITS DIRECT-ACCESS RECORD NUMBER IN
C THE  IN-CORE FREE STORAGE LIST.
C
        WRITE(6,31) ((NAME(L,K),K=1,6),L=J,JJ)
31      FORMAT(/' DUPLICATE ENTRIES:  ',5A4,5X,I4,'    FIRST ONE IS BEING
      C  DELETED'/21X,5A4,5X,I4)
        FSINDX=FSINDX+1
        FSL(FSINDX)=NAME(J ,6)
        NAME(J ,1)=KZ
        NUMDEL=NUMDEL+1
        DEL(NUMDEL)=NAME(J,6)
        CON(NUMDEL)=NAME(JJ,6)
        DELET1 = NUMDEL
        GO TO 105
C
C
6       IF(NAME(J,K).EQ.KBLA)  GO TO 2
        DO 7 K=1,6
        TEMP=NAME(J,K)
        NAME(J,K)=NAME(JJ,K)
        NAME(JJ,K)=TEMP
7       CONTINUE
        EXFLAG=EXFLAG+1
        GO TO 2
C
C
C
8       IF(EXFLAG.LT.1)  GO TO 10
 9      I=I-1
        IF(I.GT.1) GO TO 1
10      RETURN
        END


        SUBROUTINE SORT2(*,*)
C
C THIS SUBROUTINE SORTS THE IN-CORE IDNUMBER INDEX TABLE
C IT ALSO CROSS-CHECKS  WITH SORT1 FOR DUPLICATE ENTRIES.
C IF DELETIONS BY SORT1 DO NOT CORRESPOND EXACTLY WITH DELETIONS BY
C SORT2 THEN THE UPDATE WILL BE SCRAPPED AND ALL THE TABLES AND FILES
C WILL BE RESTORED TO THE STATE THEY WERE IN PRIOR TO THE START OF
C THE EXECUTION OF THE COMMAND WHICH  CAUSED THE DISASTER.
C
C
```

```
      IMPLICIT INTEGER(A-W),LOGICAL(X-Z),REAL($)
      DIMENSION IDNUM(500,2),FSL(19),NAMKEY(500,6),CON(5),DEL(5)
      LOGICAL TROUBL
      COMMON /TABLES/NAMKEY,IDNUM,FSL,DEL,CON
      COMMON /PARAM/TOTAL,MAXID,FSINDX,TEMP,NUMDEL
      COMMON /CROSS/DELET1
C
C
      TROUBL = .FALSE.
      IF(NUMDEL.EQ.0) DELET1=0
      DELET2 = 0
1     I=TOTAL
2     J=0
      EXFLAG=0
3     J=J+1
      IF(J.GE.I) GO TO 9
      JJ=J+1
      IF(IDNUM(J,1)-IDNUM(JJ,1)) 3,4,7
C
C
  4   IF(IDNUM(JJ,1).EQ.999999999) GO TO 3
C
      WRITE(6,5) ((IDNUM(L,K),K=1,2),L=J,JJ)
5     FORMAT(/' DUPLICATE ENTRIES:  ',I9,5X,I4,'  FIRST ONE IS BEING
     C DELETED'/21X,I9,5X,I4)
      IDNUM(J ,1)=999999999
      DELET2 = DELET2 + 1
      DO 6 K=1,NUMDEL
      IF(IDNUM(J,2).EQ.DEL(K))    GO TO  1
6     CONTINUE
      TROUBL = .TRUE.
      NUMDEL=NUMDEL+1
      DEL(NUMDEL)=IDNUM(J,2)
      CON(NUMDEL)=IDNUM(JJ,2)
      GO TO 3
C
C
7     DO 8 K=1,2
      TEMP=IDNUM(J,K)
      IDNUM(J,K)=IDNUM(JJ,K)
      IDNUM(JJ,K)=TEMP
8     CONTINUE
      EXFLAG=EXFLAG+1
      GO TO 3
C
C
9     IF (EXFLAG.LT.1)  GO TO 11
10    I=I-1
      IF(I.GT.1) GO TO 2
11    IF (DELET2.LT.DELET1) TROUBL = .TRUE.
      IF (TROUBL) RETURN 2
      RETURN 1
      END
```

```
      SUBROUTINE STATUS (IC, COPY)
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      COMMON XTRACE, INDEX, NDATE,
     X IF(844), LAST(20), INPUT(36), LINK(34), IREC(20,40)
     X /COMAND/ LIST(16), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      DO 61 IA = 1, COPY
      IREC(IA, 5) = 0
      DO 60 IB = 1, 4
      IF (IREC(IA,IB) .EQ. 0) IREC(IA,5) = IREC(IA,5) + 1
   60 CONTINUE
   61 CONTINUE
      NSTAT = 1
   65 DO 70 IA = 1, COPY
      IF (IREC(NSTAT,5) .LT. IREC(IA,5)) NSTAT = IA
   70 CONTINUE
      IC = 7 - 3*((IREC(NSTAT,5)+2)/3)
      RETURN
      END




      SUBROUTINE TRNSFR (UNITA, UNITB, UNITC, UNITD, *)
C  MTS VERSION..... LAST UPDATED ON OCT. 14, 1970  BY J. DIMSDALE
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      COMMON XTRACE, INDEX, NDATE,
     X ENTRY(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
C
C  THIS SUBROUTINE IS USED TO TRANSFER ENTRYS FROM ONE FILE TO ANOTHER.
C
C  UNITA = MAIN FILE'S PRESENT POSITION
C  UNITB = LOAN FILE'S PRESENT POSITION
C  UNITC = TRANSFER MAIN FILE TO THIS FILE
C  UNITD = TRANSFER LOAN FILE TO THIS FILE
C
    1 READ (UNITA, 2, END=7) (ENTRY(I), I = 1, 44)
    2 FORMAT (24A1, 11I2, 8A4, I2)
      TEXT = 44 + 80 * ENTRY (44)
      STATA = TEXT + 1
      STATB = TEXT + 17
      READ (UNITA, 3, END=7) (ENTRY(I), I = 45, TEXT)
    3 FORMAT (80A1)
      READ (UNITA, 4, END=7) XNEND, (ENTRY(I), I = STATA, STATB), $COST
    4 FORMAT (L1, 4A4, 4I2, 4I7, 3I2, I6, I2, F6.2)
      COPY = ENTRY (STATA+5)
      XCOPY = COPY .NE. 0
      IF (XCOPY) READ (UNITB, 5, END=9) ((LOAN(I,J), J = 1, 37),
     X I = 1, COPY)
    5 FORMAT (24A1, I2, 4A4, 4I6, I4, I3, 2I2)
      WRITE (UNITC, 6) (ENTRY(I), I = 1, TEXT)
```

```
    6 FORMAT (24A1, 11I2, 8A4, I2 / (80A1))
      WRITE (UNITC, 4) XNEND, (ENTRY(I), I = STATA, STATB), $COST
      IF (XCOPY) WRITE (UNITD, 5) ((LOAN(I,J), J = 1, 37), I = 1, COPY)
      IF (XNEND) GO TO 1
      WRITE (UNITD, 5)
      RETURN 1
    7 WRITE (6,8) UNITA, INPUT(1), INPUT(2), (ENTRY(I), I = 1, 24)
      STOP 2
    8 FORMAT (5X, 'END OF FILE IN TRNSFR ON UNIT = ', I2, ', CALLED BY '
     X, 2A4 / 5X, 'HAVE JUST STARTED/FINISHED ENTRY:  ', 24A1)
    9 WRITE (6,8) UNITB, INPUT(1), INPUT(2), (ENTRY(I), I = 1, 24)
      STOP 3
      END




      SUBROUTINE UPDATE (*)
C
C MTS VERSION ...LAST UPDATED ON OCT. 28, 1970 BY J.DIMSDALE
C
C     THIS PROGRAM IS USED TO UPDATE THE FILES 'CATALOG',
C     'LOANS', AND 'AUTHORS'.  IT IS USED TO ADD ENTRIES
C     FOR NEW BOOKS TO THESE FILES, TO CORRECT INCORRECT
C     ENTRIES BY REPLACING THEM WITH NEW ENTRIES, AND
C     TO DELETE ENTRIES FOR LOST, STOLEN, DESTROYED, OR
C     RETIRED BOOKS.
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      DIMENSION NEW(844),OLDAUT(24),NEWAUT(24)
      DIMENSION DELETE(10,13)
      COMMON XTRACE, INDEX, NDATE,
     X OLD(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      COMMON/PARAM/DUMBY(10),INDX01,INDX02,INDX03,INDX04,INDX08,
     CINDX20,INDX21,INDX22,INDX23,INDX24,INDX25,MAXAUT
      EQUIVALENCE( OLDAUT(1),OLD(1)),(NEWAUT(1),NEW(1))
      EQUIVALENCE(SURNAM,NEW(25))
C
C
C
6     LFILE = 1
      MFILE = 1
      NFILE = 3
      OFILE = 3
      TAPE = 2
      RESTRT = 3
      INDX01 = 1
C
C FIRST 10 RECORDS OF CATALOG FILE ARE RESERVED FOR SYSTEM INFORMATION
```

```
      INDX03 = 11
      XSAVE = .FALSE.
      XDATA = .FALSE.
      INDEX = 0
      FIND(3'2000)
      WRITE(RESTRT,15)XSAVE,XDATA,LFILE,MFILE,NFILE,OFILE,TAPE,INDEX
C     CALL LOGDSK          ******REMOVED FROM MTS VERSION ***************
      WRITE(6,5)
5     FORMAT(' C) NOW CALLING SAVE....')
      CALL SAVE (&10)
10    FIND(3'2000)
      XSAVE = .TRUE.
      WRITE(RESTRT,15)XSAVE,XDATA,LFILE,MFILE,NFILE,OFILE,TAPE,INDEX
C     CALL LOGDSK          ******REMOVED FROM MTS VERSION ***************
      GO TO 19
C
C
      ENTRY CONTIN(*)
C
C ENTRY POINT FOR ATTEMPT TO RESTART  A PROGRAM WHICH CRASHED
C BEFORE THE UPDATE RUN WAS COMPLETED
C
C
      RESTRT = 3
      FIND(3'2000)
      READ (RESTRT,15)XSAVE,XDATA,LFILE,MFILE,NFILE,OFILE,TAPE,INDEX
15    FORMAT(2L1,5I2,I4)
      IF(XSAVE) GO TO 17
      WRITE(6,16)
16    FORMAT('  C) NOT POSSIBLE TO CONTINUE UPDATE.'
     1'WILL ATTEMPT TO RESTART FROM THE BEGINNING.' )
      GO TO 6
C
C
17    IF(XDATA) GO TO 22
      WRITE(6,18)
18    FORMAT('  C) SAVE OKAY, BUT MUST REPEAT DATA.')
C
C
19    NFILE = 4
      OFILE = 3
      WRITE(6,14)
14    FORMAT('  C) NOW CALLING DATA....')
      CALL DATA(&20)
20    XDATA = .TRUE.
      FIND(3'2000)
      WRITE(RESTRT,15)XSAVE,XDATA,LFILE,MFILE,NFILE,OFILE,TAPE,INDEX
C     CALL LOGDSK          ******REMOVED FROM MTS VERSION ***************
C
C
C
C
C BEGINNING SECTION OF PROGRAM THAT MERGES THE NEWENTRIES PRODUCED BY
C DATA WITH THE OLD ENTRIES CONTAINED IN SAVEFILE
```

```
C  IT HAS PROVISIONS FOR BOTH THE DELETION AND OVERLAYING OF OLD ENTRIES
C
22     WRITE(6,21)
21     FORMAT(' C) NOW STARTING MERGE SECTION.........')
C
       ENTRIS = 0
       TITLES = 0
C
C
C   READ  INFORMATION ABOUT ENTRIES TO BE DELETED, IF ANY
C     EACH DELETION CARD CONTAINS:
C                    AUTHOR SURNAME IN  COLS 1-20
C                    AUTHOR INITIALS IN COLS 21-24
C                    LC CATALOG NUMBER IN COLS 31-46
C                    ACCESSION NUMBER OF BOOK IN COLS 51-56
C
       KD=1
       XDROP = .FALSE.
       DO 2 I=1,10
       READ(8,1,END=3) (DELETE(I,J),J=1,13)
1      FORMAT(8A1,22X,4A4,3X,I7)
       LD = I
       XDROP = .TRUE.
2      CONTINUE
3      IF(XDROP) WRITE(6,4) ((DELETE(I,J),J=1,13),I=1,LD)
4      FORMAT(' * DELETIONS:'/10(5X,8A1,5X,4A4,5X,I7/))
CC
C
C RESET FILE POINTERS
       INDX01 = 1
       INDX03 = 11
       INDX25 = 1
       FIND(1'1000*INDX01)
       FIND(3'1000*INDX03)
       REWIND 4
       REWIND TAPE
C NEWRD COUNTS THE NUMBER OF ENTRIES READ FROM FILE NEWENTRIES
       NEWRD = 0
       TERM = 6
C RESET LOGICAL SWITCHES
       XRESET = .FALSE.
       XREAD = .FALSE.
       XCHAN = .FALSE.
C
C
C    READ IN OLD ENTRY FROM  THE SAVEFILE
C
    29 READ (TAPE, 30, END=100) (OLD(I), I = 1, 44)
    30 FORMAT (24A1, 11I2, 8A4, I2)
       ELMNT = 44 + 80 * OLD (44)
       READ (TAPE, 32,END=100) (OLD(I), I = 45, ELMNT)
    32 FORMAT (80A1)
       STATA = ELMNT + 1
       STATB = ELMNT + 17
```

```
      READ (TAPE, 34, END=100) XNENDO, (OLD(I), I = STATA, STATB),$COSTO
   34 FORMAT (L1, 4A4, 4I2, 4I7, 3I2, I6, I2, F6.2)
      OLDCPY = OLD (STATA+5)
      IF(OLDCPY.EQ.0) GO TO 39
      READ(TAPE, 36, END=100) ((LOAN(I,J),J=1,37),I=1, OLDCPY)
   36 FORMAT (24A1, I2, 4A4, 4I6, I4, I3, 2I2)
C
C CHECK TO SEE IF NAMES FROM CATALOG AND LOAN FILE ENTRIES  MATCH
      DO 31 I = 1, OLDCPY
      DO 31 J= 1,24
      IF(OLD(J).NE.LOAN(I,J)) GO TO 220
31    CONTINUE
C
C
C DO ANY DELETIONS NEED TO BE MADE?
39    IF(XDROP) GO TO 300
C
C HAVE ALL THE NEW ENTRIES BEEN READ FROM FILE NEWENTRIES?
   38 IF (NEWRD .GE. INDEX) GO TO 60
C
C DO WE NEED TO READ ANOTHER NEW ENTRY?
      IF (XREAD) GO TO 40
C
C  READ NEW ENTRY
C
C
      XREAD = .TRUE.
      READ (NFILE, 30, END=120) (NEW(I), I = 1, 44)
      NEWRD = NEWRD + 1
      WORDS = 44 + 80 * NEW (44)
      STATC = WORDS + 1
      STATD = WORDS + 17
      READ (NFILE, 32, END=120) (NEW(I), I = 45, WORDS)
      READ (NFILE, 34,END=120) XNENDN, (NEW(I), I = STATC, STATD),$COSTN
      NEWCPY = NEW (STATC+5)
      NUMCPY = NEWCPY
C
C  COMPARE SURNAMES AND INITIALS              SORT-MERGE
C
C
40    DO 50 I=1,24
      IF(NEW(I) - OLD(I)) 154,50,160
   50 CONTINUE
      GO TO 170
154   IF(OLD(I).EQ.KBLA)    GO TO 60
      GO TO 54
160   IF(NEW(I).EQ.KBLA)    GO TO 54
      GO TO 60
C
C
C
C IF SURNAMES SAME THEN COMPARE CATALOG NUMBERS
C
170   DO 51 J=1,4
```

```
      I = J - 1
      IF (NEW (STATC+I).NE.OLD(STATA+I))   GO TO 60
51    CONTINUE
C
C NEW COPY OR COPIES OF A BOOK ALREADY ON FILE
C SET UP SWITCHES SO THAT NEW ENTRY WILL REPLACE THE OLD ENTRY
      NUMCPY = NEWCPY - OLDCPY
      XCHAN = .TRUE.
      IF(OLDCPY.LE.0)   GO TO 54
      DO 53 I = 1, OLDCPY
53    LOAN(I,36) = NEWCPY
C
C
C
C  WRITE NEW ENTRY
C
C
   54 XREAD = .FALSE.
C
C
C MAKE AN ENTRY IN THE AUTHOR-INDEX FILE
C
      CALL INDXR(NEWAUT,NEWCPY,XRESET)
C
C INCREMENT COUNTERS
      ENTRIS = ENTRIS + 1
C
C INCREMENT POINTERS FOR THE LOANS AND CATALOG FILES
      INDX01 = INDX01 + NEWCPY
      INDX03 = INDX03 + NEW(44) + 2
C
C WRITE OUT CATALOG ENTRY TO FILE CATALOG
55    WRITE (OFILE, 28) (NEW(I), I = 1, WORDS)
   28 FORMAT (24A1, 11I2, 8A4, I2 / (80A1))
56    WRITE (OFILE, 34) XNENDN, (NEW(I), I = STATC, STATD), $COSTN
C
C
      IF(NEWCPY.NE.0) GO TO 58
      IF(.NOT.XCHAN) GO TO 38
58    IF(NUMCPY.LE.0) GO TO 63
      TITLES = TITLES + 1
C
C WRITE OUT LOAN ENTRY TO FILE LOANS
      CATNO = WORDS + 4
      WRITE (LFILE,57) ((NEW(I), I = 1, 25), (NEW(I), I = STATC, CATNO),
     X NEWCPY, NEW(STATC+6), J = 1, NUMCPY)
   57 FORMAT (24A1, I2, 4A4, 31X, 2I2)
      IF (XCHAN) GO TO 63
      GO TO 38
C
C
C  WRITE OLD ENTRY
C
C
```

```
60     CALL INDXR(OLDAUT,OLDCPY,XRESET)
C
C INCREMENT COUNTERS
       ENTRIS = ENTRIS + 1
       IF(OLDCPY.NE.0) TITLES= TITLES + 1
C
C INCREMENT POINTERS FOR THE LOANS AND CATALOG FILES
       INDX01 = INDX01 + OLDCPY
       INDX03 = INDX03 + OLD(44) + 2
C
C
       WRITE (OFILE, 28) (OLD(I), I = 1, ELMNT)
       WRITE (OFILE, 34) XNENDO, (OLD(I), I = STATA, STATB), $COSTO
62     IF (OLDCPY .NE. 0) WRITE (LFILE, 36) ((LOAN(I,J), J = 1, 37),
      X I = 1, OLDCPY)
C
C CHECK FOR LAST ENTRY IN SAVEFILE
66     IF(XNENDO) GO TO 29
       GO TO 65
63     XCHAN = .FALSE.
       IF(NUMCPY.LT.0) GO TO 66
       GO TO 62
C
C
C MERGE COMPLETED
C
65     WRITE (LFILE, 36)
64     NFILE = OFILE
       MAXAUT = INDX25 - 1
       LOANS = INDX01 - 1
C
C OUTPUT  FILE STATISTICS.
       WRITE(6,75) TITLES,ENTRIS,LOANS,MAXAUT
75     FORMAT('1   SUMMARY OF UPDATE RUN'/
      1'-    FILES NOW INCLUDE:'/
      2'0',10X,I6,' TITLES'/
      3'0',10X,I6,' CATALOG ENTRIES'/
      4'0',10X,I6,' VOLUMES'/
      5'0',10X,I6,' AUTHORS')
C
C SET UP FILE PARAMETER RECORD
       FIND(3'1000)
       WRITE(3,68) MAXAUT,TITLES,ENTRIS,LOANS
68     FORMAT(I5,3I6)
   69 WRITE (LIB, 70)
   70 FORMAT ('  C) UPDATE COMPLETED.')
   80 RETURN 1
C
C
C ERROR MESSAGES
  100 WRITE (LIB,105) (OLD(I), I = 1, 24), (OLD(I), I = 36, 43),
      X (NEW(I), I = 1, 24), (NEW(I), I = 36, 43)
  105 FORMAT (5X, 'END OF FILE ON "TAPE".  HAVE JUST READ/AM READING:'/
      X 5X, 'OLD:', 24A1, 3X, 4A4, '   NEW: '24A1, 3X, 4A4)
```

```
      STOP 2
  120 WRITE (LIB,125) (OLD(I), I = 1, 24), (OLD(I), I = 36, 43),
     X (NEW(I), I = 1, 24), (NEW(I), I = 36, 43)
  125 FORMAT (5X, 'END OF FILE ON "NFILE". HAVE JUST READ/AM READING:'/
     X 5X, 'OLD:', 24A1, 3X, 4A4, '   NEW: '24A1, 3X, 4A4)
      STOP 3
220   WRITE(6,221) OLDAUT,((LOAN(I,J),J=1,24),I=1,OLDCPY)
221   FORMAT(' ** MISMATCH OF NAMES FROM OLD CATALOG AND LOAN FILE ENTRI
     1ES:'/10X,'CATALOG ENTRY =  ',24A1/10X,'LOAN ENTRY(IES) =  ',24A1/
     210(30X,24A1))
      STOP 4
C
C
C
C THIS SECTION HANDLES DELETIONS OF ENTRIES
C
C COMPARE FIRST EIGHT LETTERS OF AUTHOR SURNAMES
300   DO 310 I=1,8
      IF(OLD(I).NE.DELETE(KD,I)) GO TO 38
310   CONTINUE
C
C COMPARE CATALOG NUMBERS
      DO 315 I=1,4
      IF(OLD(STATA+I-1).NE.DELETE(KD,I+8)) GO TO 38
315   CONTINUE
C
C COMPARE ACCESSION NUMBERS
      DO 320 I=1,4
      IF(OLD(STATA+I+5).EQ.DELETE(KD,13)) GO TO 325
320   CONTINUE
      GO TO 38
C
C IF A MATCH, THEN DISCARD THE CURRENT OLD ENTRY
325   KD=KD+1
C
C IF ALL DELETIONS COMPLETED, THEN TURN OFF THE DELETIONS FLAG
      IF(KD.GT.LD) XDROP = .FALSE.
      GO TO 29
      END
```

```
C     OBJECT FORM OF THIS PROGRAM IS STORED IN DISK FILE "UPDATE:MAIN"
C
C MAINLINE PROGRAM FOR BATCH UPDATING OF LOAN AND CATALOG FILES
C       LAST UPDATED ON MAY 12, 1970 BY J. DIMSDALE
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      COMMON XTRACE, INDEX, NDATE,
     X OLD(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
C TURN ON THE TRACE OF AUTHOR NAMES IN SUBROUTINE DATA
      XTRACE=.FALSE.
      WRITE(6,2)
2     FORMAT('  C) NOW CALLING UPDATE....')
      CALL UPDATE(&5)
5     WRITE(6,6)
6     FORMAT('  C) WHOOPEE...')
      STOP
      END
```

```
C     OBJECT FORM OF THIS PROGRAM IS STORED IN DISK FILE  "PICKUP.MAIN"
C
C MAINLINE PROGRAM FOR RECOVERY PROCEDURE WHEN A BATCH UPDATING OF
C THE LOAN AND CATALOG FILES FAILS OR IS INTERRUPTED BEFORE IT HAS
C COMPLETED THE UPDATE RUN
C PERFORMS THE SAME FUNCTIONS AS THE COMMAND 'CONTINUE' IN THE ONLINE MO
C OF OPERATION
C LAST UPDATED ON MAY 15, 1970 BY J. DIMSDALE
C
C
      IMPLICIT INTEGER (A-W), LOGICAL (X-Z), REAL ($)
      COMMON XTRACE, INDEX, NDATE,
     X OLD(844), LAST(20), INPUT(36), LINK(34), LOAN(20,40)
     X /COMAND/ LIST(22), KBLA, KYES,
     X NIN, KRD, NOUT, MON, LIB, LIBIN, LFILE, MFILE, NFILE, OFILE,TAPE,
     X ISTATU (12), LIBNM (4,10)
      WRITE(6,2)
2     FORMAT('  C) NOW CALLING CONTINUE.....')
      CALL CONTIN(&5)
5     WRITE(6,6)
6     FORMAT('  C) WHOOPEE...')
      STOP
      END
```

```
$COMMENT    BATCHENROLL                                          398
$CREATE USERS
$CREATE USERINDEX
$CREATE FREESPACE
$COPY NEWUSERS TO *SINK*
$COMMENT  NEXT COMMAND PLACES CURRENT DATE IN RECORD 3 OF CATALOG
$RUN *TIME SERCOM=CATALOG(3,3)
$RUN XEQ.ENROLL 3=CATALOG 4=USERS 5=*MSOURCE* 7=NEWUSERS
                9=FREESPACE
$LIST FREESPACE
$LIST USERS
$LIST USERINDEX
$RUN *STATUS
$SOURCE PREVIOUS

$COMMENT  BATCHUPDATE
$COMMENT  CALLING PROGRAM STRUCTURE:
$COMMENT  $SOURCE BATCHUPDATE
$COMMENT  (PLACE CARDS FOR NEW BOOKS HERE)
$COMMENT  $ENDFILE
$COMMENT  (PLACE CARDS FOR DELETIONS HERE)
$COMMENT  $ENDFILE
$COMMENT  NEXT COMMAND PLACES DATE & TIME INTO
$COMMENT  LINE NO. 4 OF THE FILE CATALOG
$RUN *TIME SERCOM=CATALOG(4,4)
$CREATE NEWBOOKS TYPE=SEQ
$CREATE DELETIONS TYPE=SEQ
$CREATE NEWENTRIES TYPE=SEQ SIZE=1000
$CREATE SAVEFILE TYPE=SEQ SIZE=10000
$COPY *MSOURCE* TO NEWBOOKS
$COPY *MSOURCE* TO DELETIONS
$LIST NEWBOOKS
$LIST DELETIONS
$RUN XEQ.UPDATE 0=AUTHORS 1=LOANS 2=SAVEFILE 3=CATALOG
                4=NEWENTRIES 7=NEWBOOKS 8=DELETIONS
$LIST AUTHORS
$LIST LOANS
$LIST CATALOG
$LIST NEWENTRIES
$LIST SAVEFILE
$SOURCE PREVIOUS

$COMMENT    LIBRARY
$COMMENT      LIBRARY PROGRAM LOADER.....SEPT. 25, 1970
$SET ECHO=OFF
$COMMENT  NEXT COMMAND PLACES CURRENT DATE IN RECORD 3 OF CATALOG
$RUN *TIME SERCOM=CATALOG(3,3)
$RUN XEQ.LIBRARY 0=OVERDUES 1=LOANS 2=AUTHORS 3=CATALOG
                 4=USERS 5=*MSOURCE* 7=OFFLINE.I/O
                 8=USERINDEX 9=FREESPACE
$RUN *STATUS
$SET ECHO=ON
$SOURCE PREVIOUS
```

```
$COMMENT    PICKUP
$COMMENT  CALLING PROGRAM STRUCTURE:
$COMMENT     $SOURCE PICKUP
$COMMENT  NEXT COMMAND PLACES DATE & TIME INTO
$COMMENT  LINE NO. 4 OF THE FILE CATALOG
$RUN *TIME SERCOM=CATALOG(4,4)
$RUN XEQ.PICKUP 0=AUTHORS 1=LOANS 2=SAVEFILE 3=CATALOG
                4=NEWENTRIES 7=NEWBOOKS 8=DELETIONS
$SOURCE PREVIOUS


$COMMENT     XEQ.ENROLL
$CONTINUE WITH BLKDAT4 RETURN
$CONTINUE WITH ENROLL:MAIN RETURN
$CONTINUE WITH SETUP RETURN
$CONTINUE WITH ENROLL RETURN
$CONTINUE WITH SORT1 RETURN
$CONTINUE WITH SORT2 RETURN
$CONTINUE WITH CHECK RETURN
$CONTINUE WITH BINS1 RETURN
$CONTINUE WITH BINS2 RETURN
$CONTINUE WITH DAY RETURN


$COMMENT    XEQ.LIBRARY
$CONTINUE WITH BLKDAT4 RETURN
$CONTINUE WITH MAINLIB RETURN
$CONTINUE WITH BINS1 RETURN
$CONTINUE WITH BINS2 RETURN
$CONTINUE WITH BINS3 RETURN
$CONTINUE WITH BRING RETURN
$CONTINUE WITH CHANGE RETURN
$CONTINUE WITH CHECK RETURN
$CONTINUE WITH DAY RETURN
$CONTINUE WITH DEMAND RETURN
$CONTINUE WITH ENROLL RETURN
$CONTINUE WITH ALTER RETURN
$CONTINUE WITH ENQUIR RETURN
$CONTINUE WITH IDENT RETURN
$CONTINUE WITH LIST RETURN
$CONTINUE WITH QUICK RETURN
$CONTINUE WITH RESERV RETURN
$CONTINUE WITH SEARCH RETURN
$CONTINUE WITH SETUP RETURN
$CONTINUE WITH SORT1 RETURN
$CONTINUE WITH SORT2 RETURN
$CONTINUE WITH STATUS RETURN
```

```
$COMMENT    XEQ.PICKUP
$CONTINUE WITH BLKDATA3 RETURN
$CONTINUE WITH PICKUP.MAIN RETURN
$CONTINUE WITH UPDATE RETURN
$CONTINUE WITH SAVE.BATCH RETURN
$CONTINUE WITH DATA RETURN
$CONTINUE WITH INDXR RETURN
$CONTINUE WITH TRNSER RETURN

$COMMENT     XEQ.UPDATE
$CONTINUE WITH BLKDATA3 RETURN
$CONTINUE WITH UPDATE:MAIN RETURN
$CONTINUE WITH UPDATE RETURN
$CONTINUE WITH SAVE.BATCH RETURN
$CONTINUE WITH DATA RETURN
$CONTINUE WITH INDXR RETURN
$CONTINUE WITH TRNSER RETURN
```

B29990